NETWORK BASED FINGERPRINTING TECHNIQUES FOR INDUSTRIAL CONTROL SYSTEMS

A Dissertation
Presented to
The Academic Faculty

By

David Formby

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2017

NETWORK BASED FINGERPRINTING TECHNIQUES FOR INDUSTRIAL CONTROL SYSTEMS

Approved by:

Dr. Raheem Beyah, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. John Copeland, Co-Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Henry Owen
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Yusun Chang
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Alenka Zajic
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Saman Zonouz
School of Engineering
Rutgers University

Date Approved: August 24, 2017

To my parents who have always encouraged me to do my best, and to all the family and friends who provided love and support every step of the way to achieve it.

ACKNOWLEDGEMENTS

I would like to express my greatest appreciation to Dr. Raheem Beyah for serving as my advisor for the past five years. He gave me the freedom to pursue my own path when I wanted, and the necessary guidance when I needed it. I look forward to many more years of our collaboration.

Next, I would like to thank Dr. John Copeland for serving as my co-advisor and for first getting me interested in security. When I took his network security class at the beginning of my grad school career I immediately knew what area of research I wanted to be involved in, and he gave me my first opportunity with a summer research project the very next semester.

I would also like to thank Dr. Henry Owen, Dr. Yusun Chang, Dr. Alenka Zajic, and Dr. Saman Zonouz for serving as the other members of my committee, and for all of the support and guidance they provided.

My appreciation goes out to Dave Robinson and Deni Dorsey for all of the knowledge, experience, and support they have given me over the course of my research. This work would not be possible without them.

This dissertation could also not have been possible without the direct and indirect support from all of my labmates over the years, so my thanks goes out to Sam Litchfield, Celine Irvine, Paul Wilson, Qinchen Gu, Christian Bayens, Srikar Durbha, Shouling Ji, Xiaojing Liao, Preethi Srinivasan, Chris Wampler, and Sang Shin Jung.

Finally, I could not have survived this journey without the continuing love and support from all of my friends and family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Fingerprinting techniques operating over the network are proposed to identify various aspects of industrial control systems (ICSs) including software, hardware, and physical devices. First, a detailed traffic characterization is performed on several power substation networks to guide the development of the techniques. Round trip times for the resource-starved embedded devices were observed to be heavily clustered based on device type no matter how large the physical distance between them, suggesting they were largely based on processing time. This insight led to the development of cross-layer response time fingerprinting to passively identify device types based on the processing time between TCP level acknowledgments and application layer responses, with classification accuracy reaching 99% on real-world substation traffic. Complementing these techniques by addressing a different aspect of ICS networks, methods are developed to fingerprint the physical devices of the ICS. Previous work on physical fingerprinting is extended to improve relay classification from 92% to 100% and extend the scope of the methods to valves, motors, and pumps. Building on the idea behind the cross-layer response time methods, techniques are explored that expand the scope to general programmable logic controllers by generating program fingerprints from the execution times of control programs. The security of this technique is enhanced by the addition of proof-of-work functions to provide an upper bound guarantee that no additional instructions are being executed in the program. Performance of all the fingerprinting techniques are discussed with respect to their potential to contribute to a holistic, ICS-specific intrusion detection system.

CHAPTER 1
INTRODUCTION

In recent years, the confluence of increasingly complex control systems and advancing information technology (IT) has resulted in a new class of networks, cyber-physical systems. These networks in the form of industrial control systems (ICS) and supervisory control and data acquisition (SCADA) systems, are used for a variety of applications ranging from delicate manufacturing processes to critical infrastructures, such as power or water distribution where safe and reliable operation is paramount. The protocols running on these networks were often originally developed for dedicated serial communication lines and later adapted for TCP/IP networks. Although these systems now communicate over networking protocols familiar to the IT world, the true effect of the merging of these two technologies has yet to be thoroughly studied, and as a result, most research in the area of cyber-physical systems is based on artificial testbeds and theoretical assumptions that are not always accurate. One of the areas with the most profound lack of understanding as these systems migrated to TCP/IP communication was the security implications. With ICS networks in their previous form relying completely on local, dedicated serial links, the only security concerns were physical access to the building or devices. As the boundary between the control systems and the corporate network began to erode for efficiency reasons, the understanding of the new security concerns lagged behind and has only recently become a popular subject.

Unfortunately, there are several obstacles preventing solid security from being implemented in ICS networks. First, there is little public knowledge on how these networks behave in the real world. Given the nature of what these systems are actually controlling, detailed knowledge of how they truly behave is necessary for both improving operation performance and developing defenses against a new class of cyber attack that could result in physical harm to both equipment and personnel. Intrusion detection algorithms that per-

form well in lab experiments and are based on assumptions about the target network do not always translate well to real-world ICS networks where the nodes are comprised of a heterogeneous mix of devices of varying ages and the network architecture and configuration are not as clean and straightforward. Without published data on the size and intensity of bandwidth on these networks, security researchers designing these new intrusion detection systems cannot know what kinds of throughput their algorithms must handle.

Secondly, attacks on ICS networks differ from traditional attacks on IT both in their methods and goals. Rather than attempting to steal valuable data, an attacker intruding on an ICS network can theoretically inject false data or commands and drive the system into an unsafe state. Example consequences of such an intrusion can range from widespread blackouts in the power grid [1] to environmental disasters caused by tampering with systems carrying water, sewage [2], oil, or natural gas. These false data and command injections could be thwarted using strong cryptographic protocols that provide integrity and authentication guarantees, but in ICS networks it is often infeasible to upgrade legacy equipment to provide them due to lack of processing power, devices being in remote locations, and the critical nature of the systems that must be online at all times. In fact, some vendors do not even support the functionality of upgrading devices to install critical patches. When our previous research found vulnerabilities in several power system devices and they were reported to ICS-CERT, the resulting official advisory for one of the products stated that "There is no method to update [GE] Hydran M2 devices released prior to October 2014 [3]." Since adding cryptography to resource limited devices and keeping them patched is infeasible and sometimes just impossible, alternative methods such as fingerprinting must be used to provide security and intrusion detection.

Finally, standard approaches to intrusion detection and fingerprinting do not translate well to ICS networks. Even though device fingerprinting is a well-studied topic with several solutions already proposed, none of them until recent work [4] are properly suited for the ICS environment. Typical active fingerprinting techniques, such as nmap [5], can achieve

high accuracy detection of operating systems and server versions, but require probing the network with specially crafted packets. This solution is undesirable in an ICS environment where devices are performing time-critical functions and administrators would rather not risk even the small chance of a port scan crashing the legacy devices and resulting in critical system downtime that includes loss of revenue and potentially life-threatening situations for affected customers such as hospitals. Therefore passive techniques are more suited, but they usually provide limited useful information or require special equipment or TCP options enabled. The first ICS fingerprinting techniques discussed in previous work proposed methods for fingerprinting both the physical actuators in the network and the embedded computing devices. However, the physical fingerprinting techniques were only demonstrated on a small subset of actuators and required several limiting assumptions.

Therefore, the overarching goal of this research was to develop novel device fingerprinting techniques to enhance intrusion detection capabilities in industrial control system networks. This is accomplished by first summarizing related work in the area in Chapter 2 and providing a detailed characterization of real-world ICS network traffic in Chapter 3. Using the insight gained from the network characterization, various fingerprinting techniques are then developed to provide comprehensive coverage of all important aspects of ICS devices, both physical and software. Previous work on physical fingerprinting is extended in Chapter 4 followed by methods for fingerprinting embedded devices based on their processing power. This idea is then extended in Chapter 5 to fingerprinting the programs on programmable logic controllers based on their execution times and providing more robust guarantees by incorporating proof-of-work functions. Finally, conclusions from all different aspects of this research are summarized in Chapter 6 with directions for continuing research.

CHAPTER 2
RELATED WORK

Previous work related to this research, while limited, can be divided into two broad categories of network characterization and ICS-specific intrusion detection and fingerprinting.

2.1 ICS Network Characterization

Most areas of networking research depend on a foundation of knowing the target networks' behavior and traffic patterns, therefore it is crucial that detailed characterizations are regularly conducted. These characterizations can provide insight on how to run a more efficient and reliable network, help create more accurate simulation models, enable OS or browser fingerprinting based on variations in protocol implementations [5] [6], and aid in the design of more precise and effective anomaly based intrusion detection algorithms. Unfortunately, until recently there has been little research published on the characterization of power system networks and industrial control system traffic in general.

However, there has been significant research into various aspects of Internet traffic that provides a solid foundation for this research. One of the first works to study Internet traffic at a high level was achieved by Vern Paxson in 1999 when he published a study on the end-to-end behavior of bulk TCP transfers across nodes on the Internet. His results included observations on packet loss, out of order deliveries, bottleneck bandwidth, and packet replication, while offering keen insight into the causes of any abnormal behavior seen [7]. The next major milestone in traffic characterization was in 2003 with the proposal of using GPS synchronization to produce more detailed timing analysis. The study also found that the traffic content flowing through the Internet had shifted to being a majority of file sharing and media streaming as opposed to simple web sites [8].

One of the primary purposes of characterization of TCP traffic has always been to study

possible techniques for congestion control and more efficient use of bandwidth. As the applications used by the average consumer constantly demand more and more bandwidth, this is still as important as ever. However, this problem does not apply to ICS networks where bandwidth requirements are at fixed low levels and nodes consist almost exclusively of embedded devices. Even though we are becoming more and more reliant on intelligent electronic devices that control physical entities, there have been very few studies on how these specific types of networks perform. One of the earlier related works studied the physical and link layers by simulating the effects of power substation noise on commonly used wireless protocols such as WiFi and Zigbee [9]. Several years later, an in-depth characterization of cellular machine-to-machine traffic was published by Shafiq et al., which contained a small amount of power metering traffic and focused on comparing the behavior of machine-to-machine communication with smart phone traffic over cellular networks by taking measurements of round trip time, packet loss, and temporal patterns [10].

In the specific area of industrial control systems, there has only been limited research. In 2012 a study was published that compared certain traffic characteristics of a water distribution facility with an example IT network dataset and found that the water facility did not exhibit strong diurnal patterns like the IT dataset and that the flow sizes did not quite fit the typical log-normal or Pareto distributions found in other Internet characterizations [11]. Most closely related to this research was our previous traffic characterization of a power substation network in 2014 that provided an important first look into what these networks look like but only at a very high level [12]. Shortly afterward, a different study focused on characterizing widespread TCP vulnerabilities found in power grid devices in the same dataset[13]. This work differentiates itself from previous work by providing a more detailed characterization over a longer capture period, comparing behavior across multiple substations, and providing unique insight into how these observations can be used to develop ICS-specific fingerprinting techniques.

2.2 ICS Network IDS

Device fingerprinting methods are usually classified into active or passive techniques depending on whether they actively probe a device with specially crafted packets or passively monitor network traffic to develop the fingerprint.

One of the oldest and most well known fingerprinting tools, Nmap, uses active fingerprinting techniques to gather information about devices on a network [5]. By sending a series of specific requests, Nmap determines the OS and server versions running on a machine based on how the device responds. While this tool is invaluable for both pen-testers and attackers on a "normal" network, it has limited use in an ICS network where active methods are not as desirable.

For passive fingerprinting, a variety of techniques exist that provide both device type fingerprinting and individual device fingerprinting. One example is the open source p0f tool, which passively examines TCP and HTTP header fields to determine information about a client, such as OS and browser version [6]. The first attempt at formalizing methods for active and passive fingerprinting of network protocols was published in 2006, when authors used parametrized extended finite state machine (PEFSMs) to model the behavior of different protocol implementations [14]. Determining software versions is of some use, but identifying individual devices on a network based on their hardware is even more useful, which for example could be used for tracking a device across the Internet or intrusion detection. Using both passive and active techniques, Kohno et al. produced the first such work on individual device fingerprinting in 2005 by examining TCP timestamps to detect individual device clock skew [15].

Other passive fingerprinting research has focused on various timing aspects of network traffic to fingerprint devices and device types. In 2010 researchers were able to use wavelet analysis on passively observed traffic flowing through access points to accurately identify each access point [16]. The next year, another paper was published that described a method

for device fingerprinting based on models of the timing of a device's implementation of application layer protocols using Temporal Random Parametrized Tree Extended Finite State Machines (TR-FSMs) [17]. A third paper that used passive observations of network traffic timing to achieve device fingerprinting was published in 2014, and used distributions of packet inter-arrival times (IAT) to identify devices and device types [18].

Although these three papers all took different approaches to using passively observed network traffic timing to perform fingerprinting, they are all infeasible for implementation in an ICS network. The wavelet analysis approach was designed and tested only on wireless access points under heavy loads, a scenario that does not occur in ICS where wired communication is preferred for its reliability and data rates are relatively low. The method using TR-FSMs only looks at application layer behaviors and requires a large database of all possible sessions. Finally, the method using distributions of IATs requires a large number (at least 2500) of training samples to achieve accurate results, but with some devices on ICS networks being polled at an interval as large as a few seconds, this would result in unacceptably slow operation. Another technique was developed that used timing measurements of USB enumerations to fingerprint host devices [19], but this is also impractical in the ICS environment where most devices do not have USB interfaces and where it is desirable to passively fingerprint all devices on the network at once rather than driving out to remote locations to fingerprint each individual device.

Another unique approach to passive device fingerprinting relevant to this work focused on the physical layer of device communication, rather than the higher layers. Specifically, researchers were able to use amplitude and phase measurements of the signals generated by Wi-Fi radios to identify individual devices [20]. This may have been the first work to use physical measurements to fingerprint devices, but it still is not feasible in ICS networks where Wi-Fi devices are rarely used.

The first fingerprinting techniques to truly target ICS networks were developed as part of the preliminary work for this research and was published in February 2016 [4]. In this

work we proposed to fingerprint the *control* half of ICS traffic by the operation times of the physical actuators and the *data acquisition* half by the times it took the devices to process and respond to read requests. However, these techniques had several practical limitations and only studied a small subset of devices

The proposed research will overcome the limitations of the previous work on ICS device fingerprinting by providing higher accuracy results using techniques more generally applicable in the ICS environment. The first method will improve on the physical fingerprinting techniques by porting the algorithms to programmable logic controllers (PLCs), which allows for more general ICS applications and removes network architecture limitations. The second method will extend the ideas of fingerprints based on processing time to the specific case of PLC programs.

The primary use of the proposed fingerprinting techniques would be to augment existing IDS solutions, of which there is already a significant amount of previous work. One of the first attempts at tailoring IDS methods for ICS and SCADA systems was proposed by Idaho National Laboratories in 2008, and focused on monitoring traffic flows for regular patterns and understanding packets at the application layer to look for intrusions [21]. Some researchers have also approached the problem by modifying the popular Bro IDS software to perform specification based intrusion detection for common ICS protocols [22]. Others have attempted to model the states a process control system can enter and detect when a command might cause it to enter a critical state [23] [24]. These solutions are able to detect some types of attacks, but are unable to detect a class of stealthier ones called false data injection attacks. To address this, some methods have been proposed for power system state estimation [25] and for process control systems [26]. However, they are only useful in the context of power state estimation or where the process behind the control system can be accurately modeled.

Finally, while not specifically aimed at ICS devices, there have been several attempts at leveraging the limited memory and processing power on general embedded devices to

perform remote software attestation. For example, one of the first and most well known implementations of remote software attestation was presented in 2004 by Seshadri et al [27]. In this implementation, the researchers develop techniques for attesting that the software running on embedded microcontroller devices has not changed. This is accomplished by a verifier regularly issuing challenges to a remote micro-controller device performing a pseudo-random walk of the memory contents. If the checksum over the memory contents fails, or if the response takes an exceedingly long time, then the verifier knows that the program has changed. This technique was then extended to consider proxy attacks in 2010 [28]. While these were novel approaches that achieved impressive results, they are infeasible for immediate deployment in ICS networks. The approach requires the device under test to be able to read contents of the program memory, which is not made available from the control program inside PLCs. This means that ICS vendors would have to release modified versions of the firmware, which is extremely unlikely given the fact that many still struggle to even implement basic passwords on the PLCs. The fingerprinting methods proposed in this research offer novel approaches that are generic enough to be applied to most ICS networks and enable accurate detection of falsified data responses and control program modification.

CHAPTER 3

NETWORK MEASUREMENTS

3.1 Introduction

In recent years, the confluence of increasingly complex control systems and advancing information technology (IT) has resulted in a new class of networks, cyber-physical systems. These networks in the form of industrial control systems (ICS) and supervisory control and data acquisition (SCADA) systems, are used for a variety of applications ranging from delicate manufacturing processes to critical infrastructures, such as power or water distribution where safe and reliable operation is paramount. The protocols running on these networks were often originally developed for dedicated serial communication lines and later adapted for TCP/IP networks. Although these systems now communicate over networking protocols familiar to the IT world, the true effect of the merging of these two technologies has yet to be thoroughly studied, and as a result, most research in the area of cyber-physical systems is based on artificial testbeds and theoretical assumptions that are not always accurate.

Given the nature of what these systems are actually controlling, detailed knowledge of how they truly behave is necessary for improved operation performance and developing defenses against a new class of cyber attack that could result in physical harm to both equipment and personnel. Intrusion detection algorithms that perform well in lab experiments and are based on assumptions about the target network do not always translate well to real-world ICS networks where the nodes are comprised of a heterogeneous mix of devices of varying ages and the network architecture and configuration are not as clean and straightforward. Without published data on the size and intensity of bandwidth on these networks, security researchers designing these new intrusion detection systems cannot know what kinds of throughput their algorithms must handle. Additionally, detailed understanding of real-world ICS networks can lead to more accurate computer simulations and more efficiently and robustly designed networks in the future.

To date there have been very few works that attempt to provide a detailed characteri-

zation of ICS networks, primarily due to the culture of the industry being so resistant to change and the issue of security only recently gaining popularity in the area. For this work, we were given rare access to several live power substation networks and monitored the traffic over an extended period of time to answer important questions about the characteristics of real-world ICS networks. Specifically, the primary goal of this characterization was to examine the accuracy of common assumptions that are made about ICS networks and highlight any unusual behavior including observable security issues. A secondary goal was then to determine, based on the observed traffic, whether TCP is well-matched to ICS networks and whether any modifications could be made to improve the resiliency of remote field devices and efficiency of their low powered hardware. The major contributions of this chapter can be summarized as follows:

- The most in-depth traffic characterization of real-world live power substation networks to date
- A study of the stability of this characterization over 2.5 year's time and minor changes in architecture
- A comparison of behavior across multiple substations
- The discovery of several abnormal behaviors in the observed SCADA protocol stacks, configurations, and real-time software
- Suggestions for modifying the TCP protocol to fit the needs of the ICS environment

The remainder of this chapter is organized as follows. Background information on SCADA networks is provided in Section 3.2 and an explanation of the experimental setup used in this research is given in Section 3.3. Section 3.4 describes in detail the behavior of one substation network over time and Section 3.5 compares the same behavior across multiple substations. The implications of our findings are discussed in Section 3.6 and finally our conclusions and next steps were summarized in Section 3.7.

3.2 Background

In order to better understand the differences between ICS networks and IT networks, some background information on SCADA and power grid networks is required. First, we describe the general operation of a SCADA system in a power distribution network to provide context about where the network traffic capture was taken. Figure 3.1 illustrates the typical hierarchy where the control center (CC) communicates primarily with the remote terminal unit (RTU) in each substation, and the RTU acts as a middle man between the control center and intelligent electronic devices (IEDs) by collecting data from the IEDs in the field, summarizing the data and reporting it back to the CC when requested. When control operations are required, the CC sends the command to the RTU which forwards it on to the correct IED. The most common SCADA protocols used to perform this kind of communication include Modbus, GOOSE/IEC 61850, and DNP3, all of which collect data in slightly different ways. Modbus, the oldest of the three, relies solely on regular polling of measurement data, while DNP3 can operate in polling mode or spontaneous reporting mode. Finally, GOOSE operates on a publisher-subscriber philosophy by multicasting event data throughout the network.

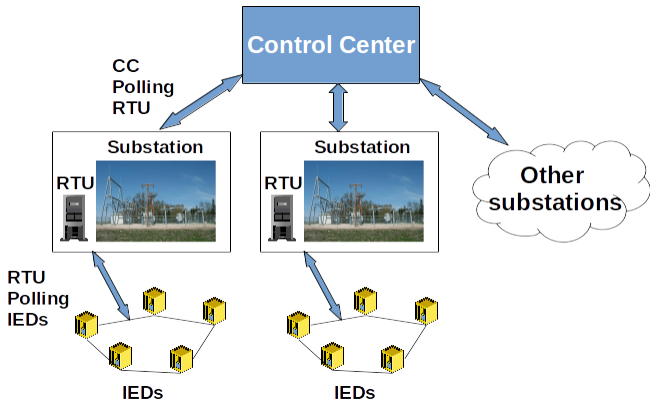


Figure 3.1: Typical SCADA hierarchy in a power distribution substation

The primary protocol used in this research was DNP3. The DNP3 protocol was originally developed for serial communication lines so it has its own complex protocol stack

illustrated in Figure 3.2, that either sits directly on a serial line or is encapsulated by the TCP/IP protocol suite. It is important to note that no matter the physical media on which the protocol is deployed, the DNP3 stack still views communication as it would appear on a serial line, as a stream of data. Therefore, the DNP3 data link layer performs frame delimiting, error checking, and optional acknowledgments for reliable transmission. The transport pseudo-layer primarily handles fragmentation of application layer fragments that are too large for the maximum link layer size. Finally, the application layer handles larger sized fragmentation, optional acknowledgments, and a wide variety of flexible functions for SCADA system operation including data collection and control. DNP3 data collection can be achieved through field devices spontaneously reporting important events, or as in the case of this research, the DNP3 master implementing a polling schedule for all devices [29].

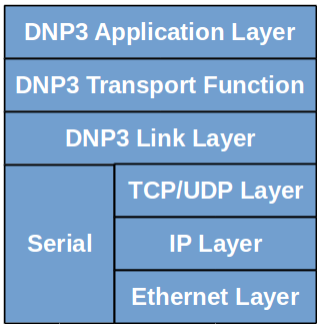


Figure 3.2: DNP3 Protocol Stack

3.3 Experimental Setup

The datasets used in this research were captured at medium-voltage distribution substations over the span of two and a half years, with roughly a year gap. During that gap, small changes were made to the configuration and architecture of the network which allows us to study how those changes affected the traffic measurements.

Figure 3.3a illustrates the architecture from which the first dataset was obtained. Under this architecture, each substation has its own separate LAN, communication between the

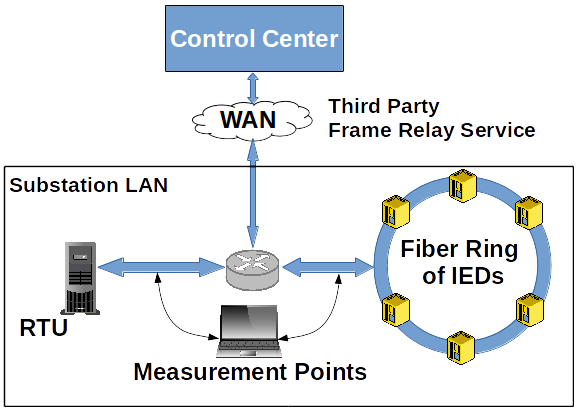
Table 3.1: Description of Datasets

	Start	End	Size [GB]	Nodes
A1	Sept. 2013	Feb. 2014	21.7	204
A2	Jan. 2015	Aug. 2015	146.4	317
A3	Aug. 2015	April 2016	147	517
B	10 Aug. '15	11 Aug. '15	0.34	118
C	April 2016	May 2016	5.7	129
D	April 2016	May 2016	11.7	167

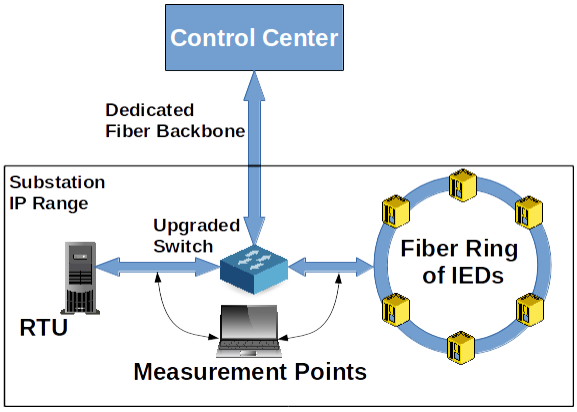
CC and RTU was over a third-party frame relay service, and DNP3 polling intervals were much slower than in the second dataset. The traffic was captured in the substation close to the RTU so all communication between the CC, RTU, and IEDs in the field could be monitored. The capture setup in Figure 3.3b is different in that all substations were moved to the same LAN, a new switch was installed that allowed port mirroring, communication between the CC and RTU used a dedicated fiber backbone, and the DNP3 polling intervals were much faster than the first dataset. Table 3.1 summarizes the basic statistics about the datasets, where datasets A1, A2, and A3 are taken from the same substation and the others are taken from separate substations on the same network. The change from using a third party frame relay service to having a private fiber backbone would theoretically increase the speed and reliability of the communication between the CC and each substation. Furthermore, while data on the old router's processing and switching delay is unavailable, it is relatively safe to assume that the brand new, higher end switch is able to process packets at a higher speed reducing the switching delay for packets on the network. Finally, the change from having each substation on its own LAN to logically combining them on the same LAN should have little effect on the traffic observed at the specified measurement points, except for potentially seeing some traffic from nearby substations.

3.4 Stability Over Time

The first half of this characterization studies the evolution of a substation's behavior over a long period of time using datasets A1 and A2 to determine how stable the measurements



(a) A1 Network Architecture



(b) Network Architecture for All Other Datasets

Figure 3.3: Experimental Setup

were.

3.4.1 High Level Behavior

To first study the network traffic at the highest level of abstraction, patterns in metadata such as timing and bandwidth usage were examined.

Traffic Volume. As explained in Section 3.1, the two primary functions of any ICS or SCADA system are data acquisition and control. Consequently, the networks are assumed to have very clear communication relationships where the bulk of the traffic is generated from field devices regularly reporting data to the master and the master occasionally sending commands as needed. Although it varies by the exact SCADA protocol used, this is especially true with the DNP3 protocol where there are strict master/slave relationships

Table 3.2: Bandwidth statistics averaged over ten-second samples

	A1	A2
μ [kbps]	11.2	58.8
σ [kbps]	7.7	860.2
Max [kbps]	1925	278931
Min [kbps]	1.11	52.1

Table 3.3: Average Packet Sizes

	A1	A2
μ [bytes]	89.9	96.5
σ [bytes]	91.3	107.6

and the IEDs never communicate with each other. To test whether the bandwith is low and stable, as would be expected from polling traffic, bandwidth samples were taken every ten seconds and summarized in Table 3.2.

The results confirmed that indeed, the network uses very little bandwidth on average at 11kbps and 58kbps for A1 and A2 respectively of the 100Mbps links and with little variation. The bandwidth usage increased between datasets due to a configuration change that increased the polling frequency of devices for measurements. Another interesting observation is the maximum bandwidth usage in both datasets appear to be caused by temporary switching loops and subsequent broadcast storms.

As a result of the constant polling for measurements, and the device often having no signficant events to report, the average packet sizes remain rather small, as described in Table 3.3.

Regularity of Traffic. Popular ICS protocols including DNP3 and Modbus rely on regular polling of measurement data to ensure that the system is operating as intended. DNP3 in particular uses a combination of faster event polls to monitor for exceptions, such as voltages crossing a certain threshold, and slower static polls to monitor the exact status of all measurements over time. These polls are initiated by devices running real-time operating systems and are often configured with the real-time control constraints in mind, i.e. certain measurements have to be updated so often in order maximize efficiency or respond

to emergencies in time. Therefore, one might assume that the polling intervals would be very regular with low jitter, but this was not the case. Figure 3.4 illustrates the event and static polling intervals for all devices in datasets A1 and A2. Note that all polls were initiated by the RTUs running VxWorks, a popular real-time operating system that provides hard real-time guarantees. Minimum and maximum polling intervals varied drastically due to broadcast storms and devices going silent, and even during normal operation polling intervals had standard deviations on the order of several seconds despite the real-time nature of the RTU. To verify that this behavior was not caused by network issues, polling intervals were also calculated by the TCP timestamp ticks in the poll requests coming from the RTU, which were generated by the real-time operating system. Indeed, Figure 3.5 illustrates how the poll requests being generated by the software running on the real-time operating system in the RTU exhibits similar irregularities.

To study another aspect of the regularity of the traffic, the inter-arrival times of packets at the capture point were measured and are shown in Figure 3.6. The distributions from both datasets appear to be approximately the same, and the most interesting feature is small periodic spikes every 17ms that dampen exponentially. While the cause for these spikes is still unclear, we speculate that it is most likely originating in the RTU since it initiates all communication with the IEDs. One possible explanation is the use of a very coarse grained timer to decide when to send the next polling request, but other explanations are just as likely to be the true cause. It is also interesting to note that the traffic intensity of the network is so low that inter-arrival times occasionally reach as high as one second.

Availability. One interesting aspect that differentiates ICS networks from traditional IT networks is the consequences of node down-time. If a set of nodes is unavailable for a period of time in an IT network, either due to network issues or device issues, revenue and productivity will be lost but no physical harm will be done. However in the ICS environment, every minute that a node is down increases the risk that a critical issue that requires attention will go unnoticed. In the case of the power grid, this could be as innocuous as lost

revenue from wasteful energy production when demand has fallen, or as devastating as loss of power to parts of the grid due to increased consumption and lack of available power.

Figure 3.7 illustrates the idle times for individual devices with the RTU separated from the IEDs. For the first dataset, the distributions show that the RTU never appears to lose availability for long, but surprisingly some IEDs appear to go down for up to 10 days at a time. It is unclear from manual inspection of the network traffic if these devices were intentionally taken out of commission for maintenance or technical problems brought them down. Devices in the second dataset demonstrate similar behavior.

3.4.2 TCP Level Behavior

In addition to unexpected patterns in high level metadata, unusual behavior was also observed in the common network protocols that have been adapted from the IT world into use for the ICS environment, and it was found that some of the most crucial improvements to TCP that allow modern web traffic today are largely irrelevant in the context of ICS networks.

TCP Flow Duration and Size Although previous work has studied TCP flow sizes for Internet traffic [30] and for water distribution networks [11], data on power grid TCP flow sizes has yet to be published. Studies of Internet traffic suggested that flow durations have long tailed distributions and follow log-normal and Pareto distributions. The characterization of the water facility found similar results for some types of flows, but others exhibited too much variance. The general conclusion that the research was able to draw was that all types of flows observed at the water facility were positively skewed, meaning most values fell within a main body in the distribution but there were a significant number of extreme large values.

Water distribution facilities are another example of industrial control system networks not unlike the power substation networks studied here, so similar results were expected to be found. From a purely theoretical perspective, most flows were expected to be very long

lived similar to an extreme persistent HTTP connection, due to the connections staying open for the constant polling for measurements. Few shorter flows were expected to represent the occasional configuration or manual maintenance. However, results obtained from the substation network exhibited high variance due to a variety of configuration issues at the application layer and network layer, illustrated in Figures 3.8 and 3.9.

The most striking feature of the A1 distribution is the overwhelmingly large body centered around a 100 second duration. After closer inspection, the cause of this was determined to be a misconfiguration of all devices on one of the circuits in the network. The issue stemmed from the fact that the RTU was configured to poll for DNP3 event data every 100 seconds for each device in this circuit, but either the IEDs themselves or the switches in the network were configured to close the connection with a FIN handshake after 30 seconds of being idle. After acknowledging the FIN packet from the IED, the RTU never sends its own FIN to gracefully close the connection and instead waits another 80 seconds, attempts to send another DNP3 poll request and gets a TCP reset flag in response. This type of disagreement in the TCP connection termination was actually so rampant in the network that 99.8% of all flows were terminated with a TCP reset flag instead of the full graceful FIN handshake.

The second most noticeable spike (largely overshadowed in Figure 3.8 is the flow durations on the order of 10 seconds, corresponding to relatively short flows from maintenance access to various devices. Finally, the relatively large number of extremely short duration flows appeared to mostly come from a brief disruption where the switches were caught in a broadcast loop and UDP messages flooded the network causing connection issues. Other extremely short flows appear to be caused by occasional strange behavior where the RTU becomes temporarily inactive and refuses connection attempts by the control center.

When the network was revisited over a year later in dataset A2, again, large amounts of extremely short flows were a result of another broadcast storm. However, as a result of the faster DNP3 polling, the overwhelming spike at 100 seconds no longer appears because

the IEDs are never idle for 30 seconds. Instead, this reveals another strange configuration issue evidenced by the large spike at around 20 seconds. After manual inspection of the traffic, the cause of this new spike appeared to be an implementation issue at the firmware layer. At several points in both datasets, the RTU closes all connections with the IEDs for a short period and refuses connection attempts from the control center. In one of these cases, when the connection is restarted with a specific IED, the IED sends the RTU a DNP3 "Request Link Status" message. The message is acknowledged by the TCP layer at the RTU, but apparently ignored at the application layer. While normal DNP3 data collection is successfully taking place, the IED retries this message several times without getting a reply and finally closes the connection after about 20 seconds apparently with the mistaken belief that the connection is broken at the application layer. The connection is continually restarted following this same pattern throughout the rest of the dataset and is never resolved. As a result of one configuration issue being corrected and another one appearing, the total percentage of TCP flows closed by reset flags in the second dataset was 87.03%.

To compare the distributions with previous work suggesting TCP flows follow log-normal and Pareto distributions, all data was fit to both log-normal and Pareto distributions using Maximum Likelihood Estimation of the parameters. The resulting theoretical distributions were compared with the empirical ones in Q-Q plots (Figures 3.10 and 3.11) to qualitatively test for similarities, noting that the closer Q-Q plots resemble the green line $y = x$ the closer their distributions are. Also note that extreme outlier quantiles were omitted to allow for detailed examination of the main body of the distributions. Clearly, the large spikes caused by the observed undesirable behavior prevent the empirical distributions from matching any of the theoretical ones.

Round Trip Times Another notable characteristic of ICS networks is that they consist of devices in fixed locations in the same geographic area as opposed to mobile devices constantly on the move or devices communicating over large distances. Therefore, based on knowledge from traditional IT networks one might think at first that the round trip

Table 3.4: RTT Statistics

Between RTU and	μ_{A1} [ms]	σ_{A1} [ms]	μ_{A2} [ms]	σ_{A2}
Type A.1a	18.7	7.4	18.9	22.2
Type A.1b	17.9	8.12	21.1	22.6
Type A.2	18.8	7.13	23.5	47.9
Type B	1.4	1.6	58.7	188
CC	3.2	1.9	9.0	2.94

times (RTTs) would be small and consistent, since they should not be affected by changing locations, different routes over the Internet, or long propagation delays.

As explained in Section 3.3, the measurement point for our experiments was close to the substation's RTU, therefore to test assumptions about round trip times, RTTs between the RTU and field devices were measured from the dataset by recording the time between the SYN and SYN-ACK packets in the TCP handshake and the RTT between the RTU and IP addresses associated with the control center were calculated from the time between the SYN-ACK and final ACK in the three-way handshake. RTT measurements taken during a brief broadcast storm in the network were discarded due to being extreme outliers (round trip times of greater than four seconds).

As Figure 3.12 and Table 3.4 illustrate, the RTTs are neither small nor consistent. Although this may be counter-intuitive at first, it can easily be explained by the fundamental differences in the make-up of legacy ICS networks and current-day IT networks. The typical IT network consists of large numbers of relatively powerful end devices capable of processing and creating large amounts of data that must be sent over communication lines that are heavily constrained by bandwidth and propagation delay. Due to this imbalance of large load generation capability and small network bottleneck size, IT networks would quickly experience congestion collapse without implementing congestion control algorithms [31].

However, in ICS networks the imbalance leans in the opposite direction. The networks consist of low powered embedded devices where it is not uncommon to be running 16-bit microcontroller processors in the tens to low hundreds of MHz range and very limited

RAM. The communication distances at most are no bigger than a few miles resulting in propagation delays that are fractions of a millisecond. Additionally, the link bandwidths are typically over-provisioned for availability and reliability reasons (e.g., the observed network only utilized roughly 0.02% of the 100Mbps link). This imbalance suggests that the observed RTTs are largely dominated by the processing time of the end devices rather than the propagation and queuing delay across the links.

Another interesting observation from this comparison is the difference in consistency between the two types of RTTs. Even though the ICS network's traffic intensity was very light, meaning that the network switches should never have been heavily loaded, and the packets always took the same path, the RTTs were surprisingly much more erratic compared to the RTTs over the unpredictable Internet. RTTs measured over the Internet typically have standard deviations on the order of a few milliseconds. By comparison, the RTTs measured in the ICS network were widely varying with standard deviations in the tens and even hundreds of milliseconds, and it was a regular occurrence to have RTTs as large as three seconds.

In order to study whether the cause of the irregularity in RTTs originated in the network or the field devices themselves, we compare the different RTTs in Table 3.4. The last row contains statistics for communication between the RTU in the substation to IP addresses associated with the control center. This communication is carried wirelessly over a third party frame relay service across a distance (about 20 miles) that is roughly ten times the communication distance between the RTU and the IEDs over fiber. The devices in the control center that communicate with the RTU are all relatively modern PCs capable of quickly processing the TCP handshake, resulting in a standard deviation of RTTs that is expectedly smaller than any of the measured Internet RTTs or other ICS device RTTs. The fact that this link, even with its longer distance and less predictable wireless communication medium, has more stable RTTs than the links between the RTU and field devices further suggests that the embedded device processing time dominates the observed network performance

rather than the network infrastructure itself.

Finally, given such strong evidence that the RTT is largely dependent on the processing time of the embedded devices, it suggests that measuring the RTT could supply information about the identity of the device, similar to the cross layer response time fingerprinting methods proposed in [4]. Indeed, Figure 3.12 shows strong clusters for different physical device types, with Device Types A.1a and A.1b being the same hardware and Type A.2 being only a slightly different model hardware. Furthermore, these device types were not clustered by location and were quite spread out, ruling out any significant effects of propagation delay and switching delay. For example, these devices ranged from being at the substation close to the measurement point and traversing only one network switch, to being up to a couple miles away traversing upwards of thirty network switches.

With the change in architecture, upgrading of the main switch in the substation, and correction of some of the configuration issues, the RTTs appeared to be less variable, but still exhibited similar means and clusters compared to the first dataset, as illustrated in Figure 3.13 and Table 3.4, with samples taken during two broadcast storms omitted due to extreme outliers. In this dataset, the device types seem to be even more heavily clustered based on the mean and variance of the RTTs.

TCP Retransmissions Throughout the duration of a TCP conversation, both ends keep estimates of the RTT from their perspectives in order to calculate a retransmit time-out (RTO) that reduces the chances of wasting bandwidth on retransmissions of data that would have eventually arrived. In Paxson's large scale study [7] involving two captures one year apart of bulk transfers across the Internet, he found that 1% and 2% of the packets observed in the first and second captures respectively were redundant retransmissions. Of these redundant retransmissions, 44% and 17% respectively were deemed to be unavoidable due to the loss of a segment or all acknowledgments of the segment. The majority of the remaining redundant retransmissions could have been avoided using selective acknowledgments (SACK) [32], and only a small percentage (4% and 3%) could have been avoided with a

better RTO value.

Given the differences between the local power substation network and the complex Internet, the retransmission performance at the power substation network was expected to be significantly better. The combination of the low datarate (0.02% average traffic intensity), small packet size, and dedicated fiber links at the substation suggests that packet loss should be virtually non-existent and that queuing delays should be small and stable resulting in RTOs that prevent unnecessary retransmissions. However, as the results in the previous section about RTT illustrated, the low-powered embedding processing time appears to have a significant impact on the network performance including retransmissions.

As explained in Section 3.3, data was collected only at one location in the network so it is impossible to conclusively determine if a data segment and its ACK were actually received by the corresponding ends, as was the case in Paxson's study. However, given the low traffic intensity (0.02%) on the substation network it is highly unlikely that any packets were actually dropped due to congestion rather than just delayed by slow processing. To elaborate, given that the observed average packet size of 90 bytes has a transmission time of 7μs on a 100Mbps link and that the observed average time between packet arrival for the entire network is significantly greater at 65ms, the outgoing rate at which each switch can push packets on the line is much greater than the incoming rate of arriving packets. Therefore, the average packet should experience roughly zero queuing delay, packet loss should be virtually nonexistent, and almost every observed retransmission could have been avoided.

Even though the retransmission performance was expected to be significantly better on the substation network compared to Paxson's study, Table 3.5 shows only marginal improvements. The substation network had an average hourly retransmission percentage of 0.5% which reached as high as 2.86% during one hour, compared to Paxson's 1% and 2% over the Internet. Of these observed retransmissions 43.2% of them occurred *after* the acknowledgment for the original segment had already traversed the measurement point.

Table 3.5: Hourly Retransmission Statistics

	μ_{total}	σ_{total}	μ_{ACK}	σ_{ACK}
A1	0.510%	0.0951%	43.2%	5.77%
A2	1.46%	0.501%	26.6%	2.35%

Since it is unlikely that the acknowledgment was dropped anywhere in the network, the cause of these avoidable retransmissions had to be due to the sender's RTO expiring before the acknowledgment could reach it.

To examine why the overall retransmission rate was so unexpectedly high, retransmission percentages were calculated based on IP address pairs and classified by direction in the Q-Q plot in Figure 3.14a. Note that if the retransmission patterns for packets originating from the RTU were equal to those destined to the RTU, then the Q-Q plot would roughly follow the line, $y = x$. Interestingly, the retransmissions were not evenly distributed among device pairs due to some devices having high overall retransmission rates (as high as 15%) and others having much more reasonable rates of around 0.1%. The shape of the Q-Q plot falling so far below the line $y = x$ suggests that packets originating from the IEDs in the field were retransmitted at much higher rates than in the opposite direction. When studied further, it was found that the IP address pairs suffering from higher retransmission rates were located in the same electrical cabinets connected to the same switches, suggesting a possible malfunction or misconfiguration of some of the network switches. Other causes for poor performance could originate from the noisy electrical environment the devices are subjected to, despite being hardened against such conditions.

Another important observation to be made from Figure 3.14a is the imbalance between directions of the retransmissions. On closer inspection, it was found that the field devices retransmitted much more frequently to the RTU than the RTU retransmitted to the field devices, suggesting two very different measured RTTs or different calculations of the RTO.

The results from Dataset A2 in Figure 3.14b suggested similar conclusions about uneven retransmissions and offered other interesting observations as well with some IP pairs

having a majority of their conversation be retransmissions. When examined manually, this seemed to originate from some IP addresses apparently located in other substations in the network rarely showing up in normal operation but appearing and being retransmitted a number of times during the two broadcast storms. It is also interesting to note that the average retransmission rate increased to 1.45% with the new changes, most likely due to the increased bandwidth usage and more frequent queuing delays.

While it was impossible to accurately measure the RTO for all devices with only one network tap, estimates were made based on the arrival time difference between the original transmission and the subsequent retransmissions. Note that the data in Figure 3.15 excludes the extreme long tails at 64 seconds in order to offer insight into the minimum RTOs implemented by the different operating systems of the devices on the network. The original specifications for TCP did not clearly define a minimum RTO, but later RFCs recommended a minimum of 1 second or 3 seconds if no RTT measurement has been made yet [33]. This was further updated in 2011 with a modified recommendation to use 1 second if no RTT measurement has been made [34]. Both RFCs state that smaller RTOs may perform better, and it has been noted that several operating systems do use smaller values and that they can increase performance [35].

The clusters around 1 second and 3 seconds appear to primarily come from the RTU, and align with the behavior specified in the RFCs. Most of the retransmissions originating from the RTU have an RTO between 1 and 1.5 seconds, and for the short lived connections discussed in Section 3.4.2, it appears to use 3 seconds since it never has a chance to get an RTT estimate. The other clusters around 300ms and 200ms align with the observed RTOs used by modern Windows and Linux machines respectively, but the 100ms cluster and large number of RTOs below 10ms were unexpected. Since it is unlikely that these legacy devices are actually using faster minimum RTOs than more modern OSes, we suspect that packet timing compression is happening somewhere along the network or even in the slow processing of the field device itself.

Although it is difficult to see in the figure, the RTU is also observed to double its RTO as specified in the RFC until it reaches a set maximum at 64 seconds where it retries several more times before finally giving up.

Exact RTO behavior is vendor specific which means distributions from different networks will undoubtedly differ from those presented here. However, two important conclusions can be drawn from this data that *do* generalize to other ICS networks. The first is that due to the use of proprietary application specific OSes and slow update cycles, there is much more inherent diversity in the TCP implementations seen in ICS networks than a typical corporate network consisting primarily of modern Windows and Mac systems. The second conclusion reinforces what was discussed in the previous section in that the slow processing time of the embedding field devices dominates that network performance rather than the network infrastructure.

The results from Dataset A2 in Figure 3.15 produced similar results.

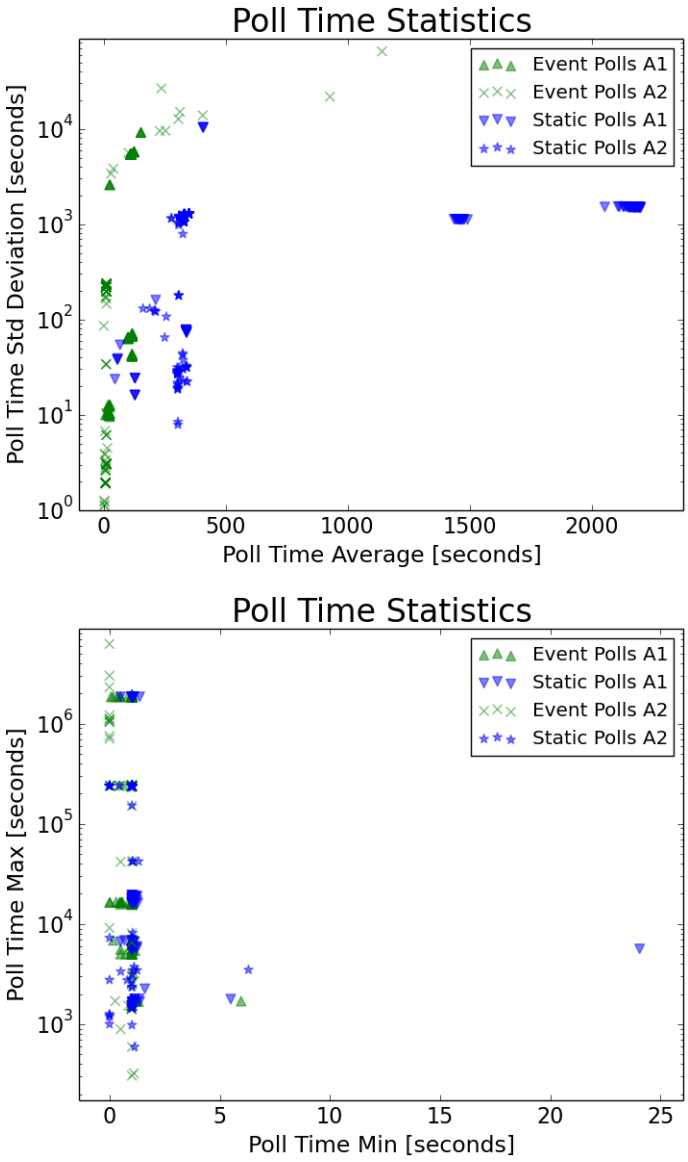


Figure 3.4: Polling Intervals

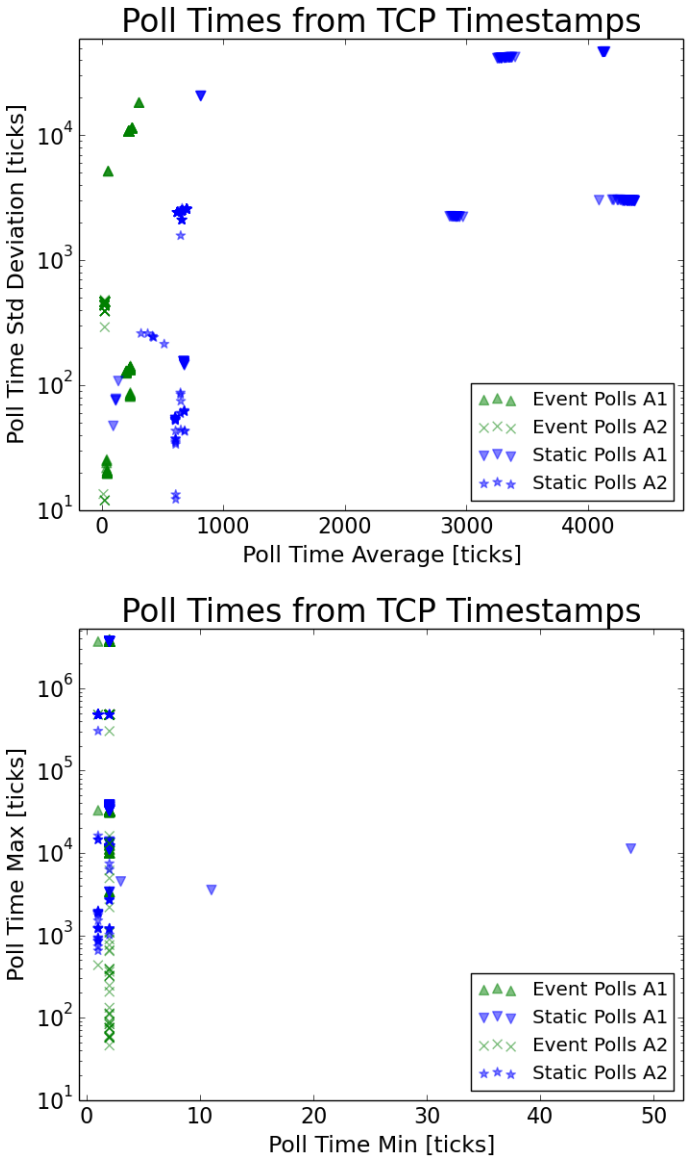


Figure 3.5: Polling Intervals from TCP Timestamps

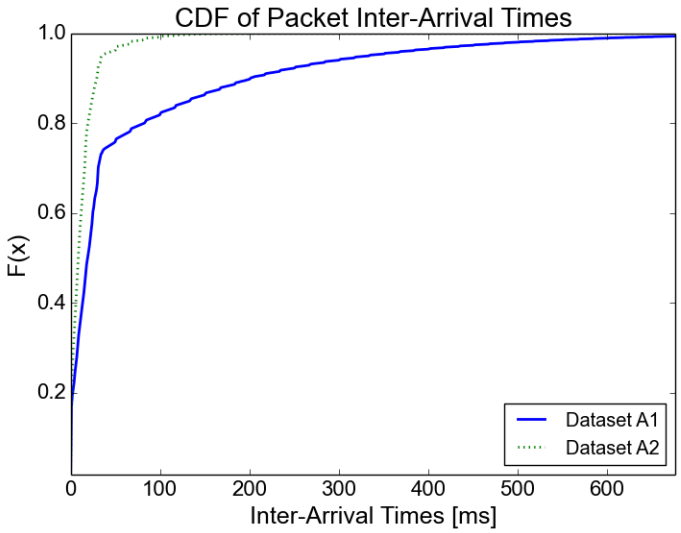


Figure 3.6: A1 and A2 packet inter-arrival times at the measurement point

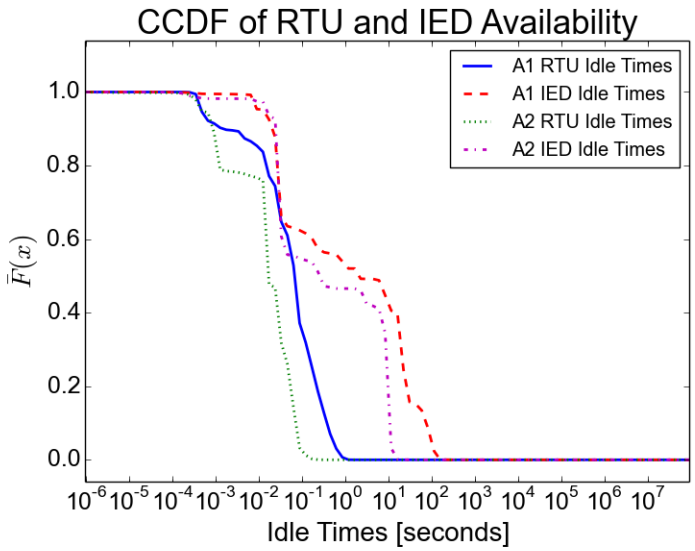


Figure 3.7: Idle Times of Devices on the Network

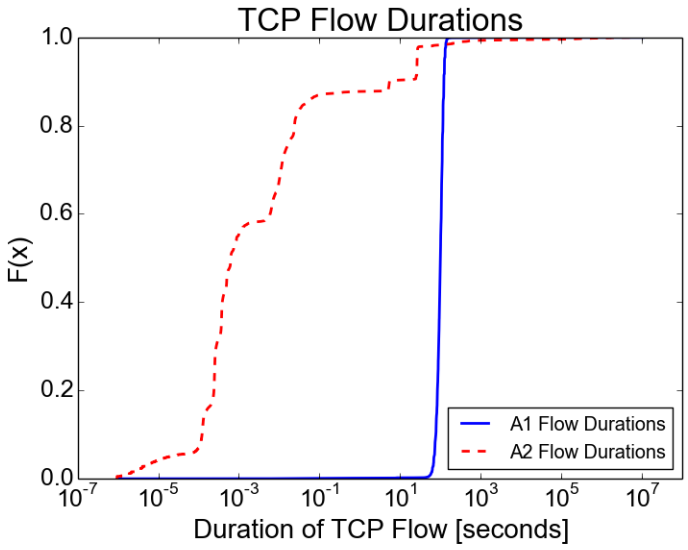


Figure 3.8: A1 and A2 TCP Flow Durations

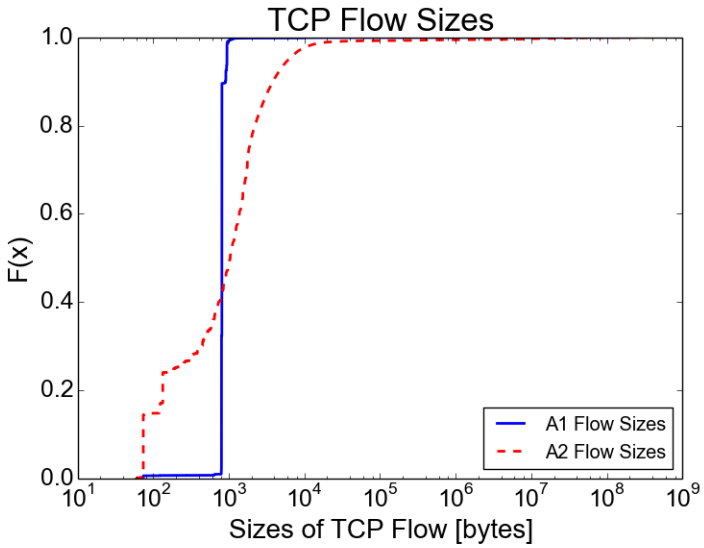
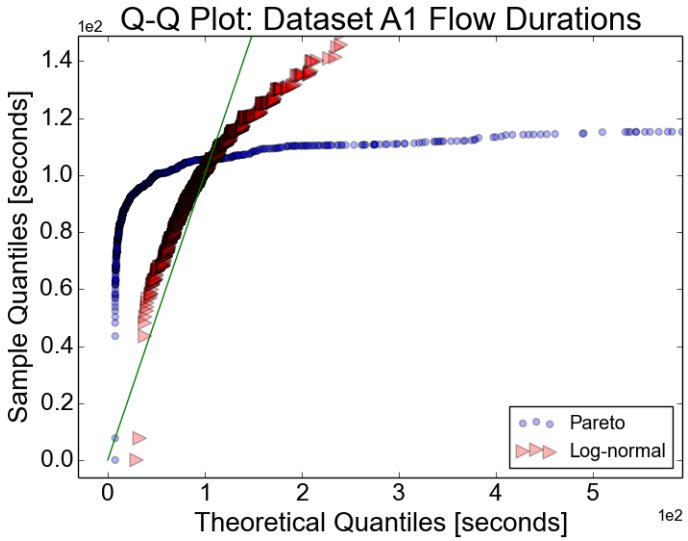
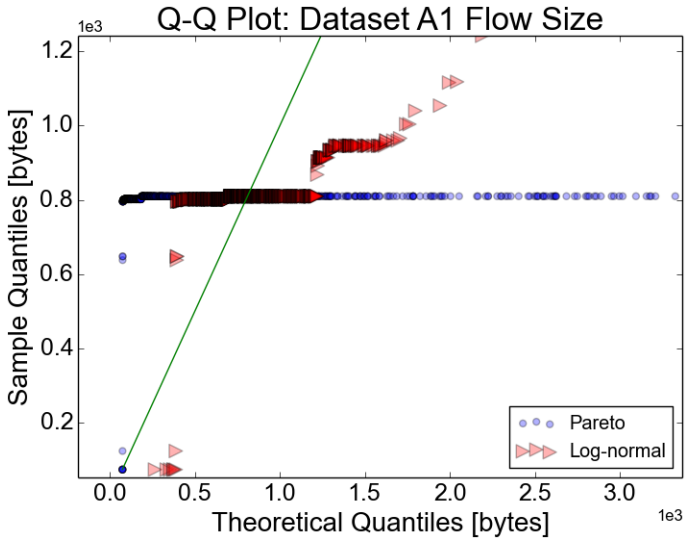


Figure 3.9: A1 and A2 TCP Flow Sizes

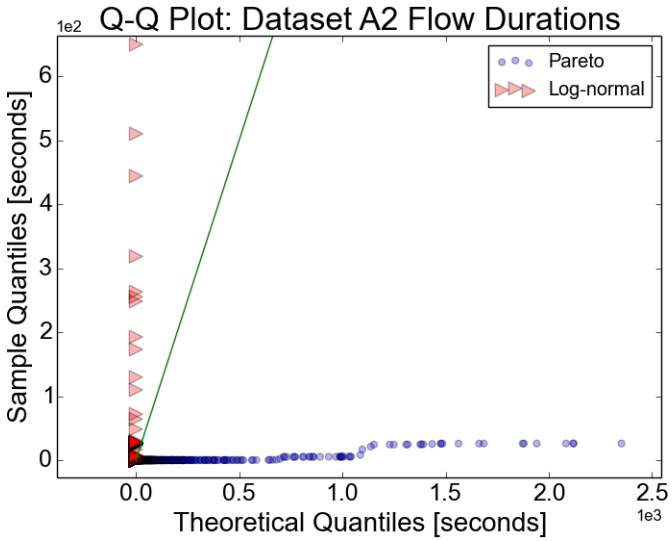


(a) A1 Flow Duration Q-Q Plots

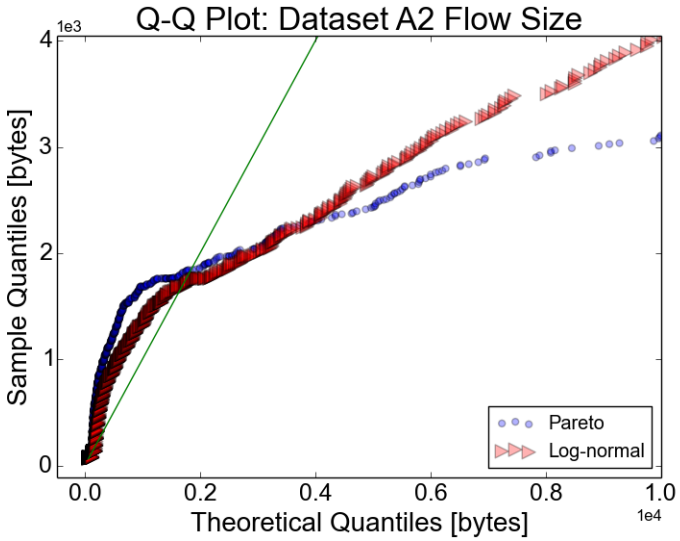


(b) A1 Flow Size Q-Q Plots

Figure 3.10: A1 Q-Q Plots



(a) A2 Flow Duration Q-Q Plots



(b) A2 Size Duration Q-Q Plots

Figure 3.11: A2 Q-Q Plots

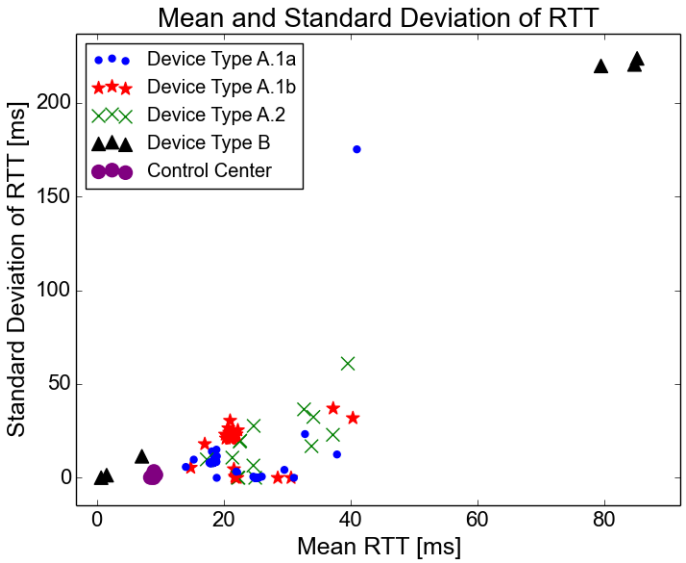


Figure 3.12: A1 mean and standard deviation of round trip times for each device

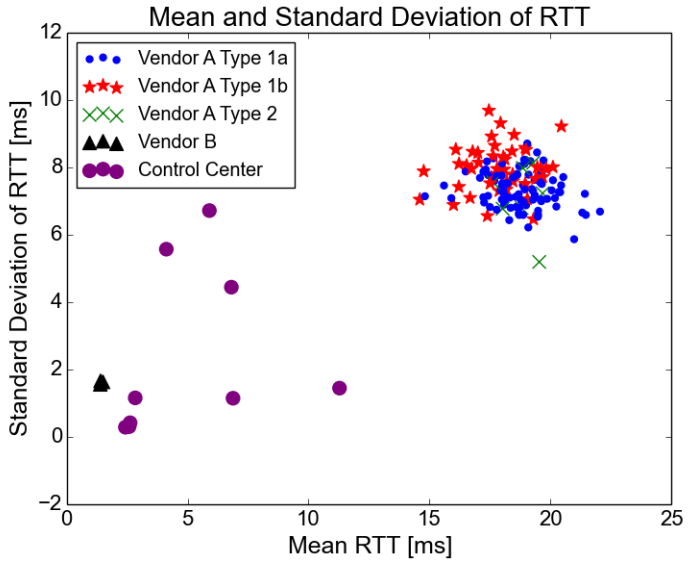
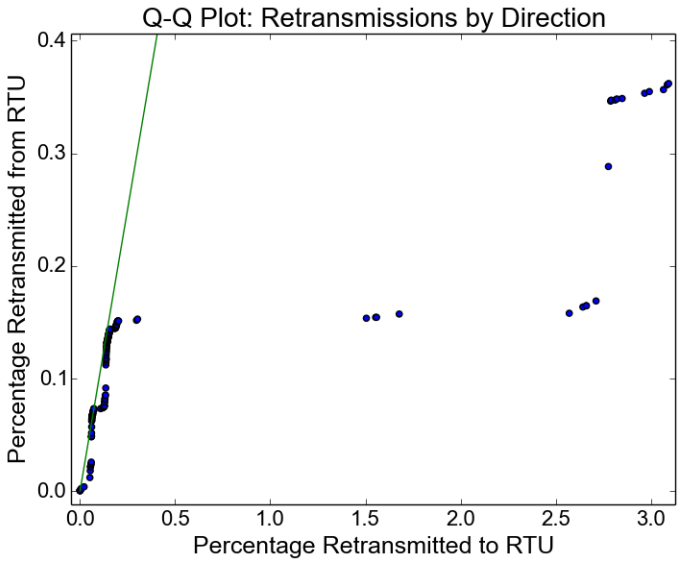
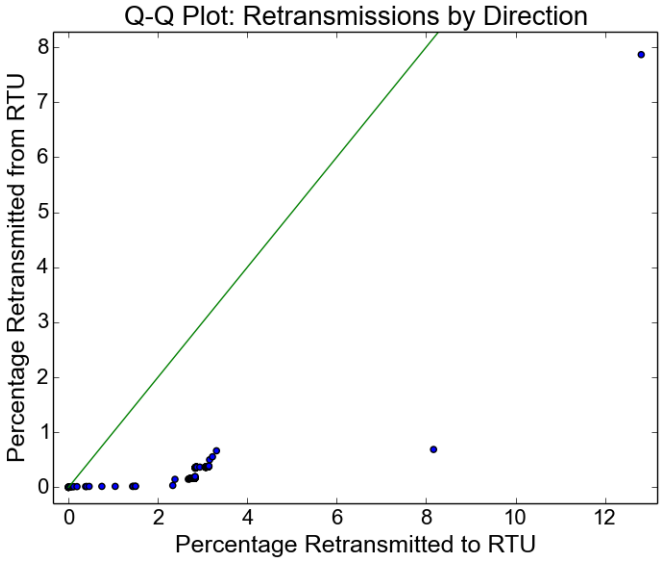


Figure 3.13: A2 mean and standard deviation of round trip times for each device



(a) Q-Q plot of A1 retransmission per flow, comparing directions to and from the RTU



(b) Q-Q plot of A2 retransmission per flow, comparing directions to and from the RTU

Figure 3.14: Imbalanced direction of retransmissions

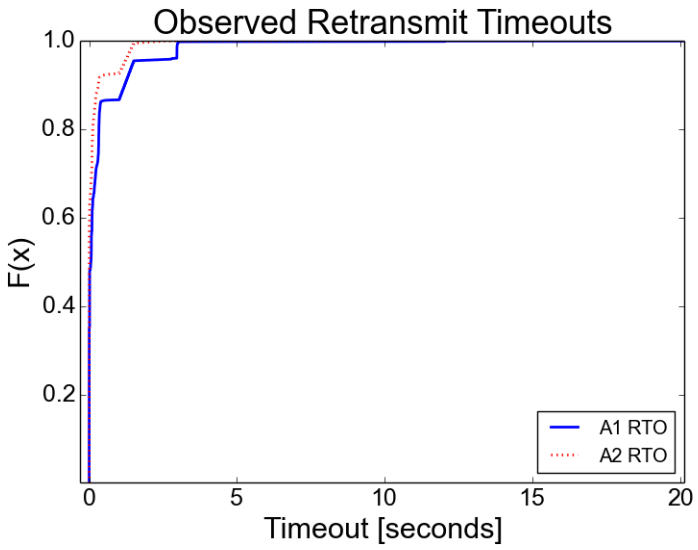


Figure 3.15: A1 and A2 RTO Behavior Less than 32 Seconds

3.5 Comparison Across Substations

After conducting the previous longitudinal study at a single substation, behavior across multiple substations was examined to determine how general the previous observations were. A third dataset from the first substation was compared with data from three other substations on the power system network.

3.5.1 High Level Behavior

Comparing differences in behavior at the highest level of abstraction, such as bandwidth usage and packet size, across substations provides insight into how active and connected in the power grid each substation is.

Traffic Volume When similar bandwidth measurements are taken as in Section 3.4.1, the new results, in Table 3.6, lead to similar observations. Again, the average bandwidth usage is low in the tens of kilobits per second, and another broadcast storm caused large spikes of traffic in A3.

Table 3.6: Bandwidth statistics averaged over ten-second samples

	A3	B	C	D
μ [kbps]	48.2	34.1	15.9	33.1
σ [kbps]	232.7	1.44	1.17	1.01
Max [kbps]	224503	46.6	45.2	103.4
Min [kbps]	0	29.2	0	25.5

The average packet sizes in Table 3.7 were again small due to the polling nature of the traffic, with the only significant differences between substations being the standard deviation of the sizes. This could be explained by the lack of broadcast storms in the shorter datasets B, C, and D, or possibly a sign of how often devices in the field have important events to report back to the master.

Regularity of Traffic Polling interval jitter was again observed across all substations from both network tap timestamps and timestamps from the RTUs at each substation, illus-

Table 3.7: Packet Sizes

	A3	B	C	D
μ [kbps]	82.2	78.1	78.5	80.0
σ [kbps]	52.9	34.9	37.1	29.2

trated in Figures 3.16 and 3.17.

When the inter-arrival times of packets at each of the substations were plotted in Figure 3.18, again the unexplained spikes at 17ms intervals are present in each one. It can also be noted that the busier substations with more devices on their networks have more short inter-arrival times simply due to the presence of more traffic.

Availability Examining the idle times of the RTUs and IEDs in Figure 3.19 reveals similar behavior with IEDs going quiet for surprisingly long periods of time. For example, from Dataset A3 a few devices were not heard from for several days while device idle times from Datasets C and D maxed out around 10 minutes of idle time.

3.5.2 TCP Level Behavior

Measurements made at the TCP level across substations revealed more evidence of configuration issues and confirmed observations made about the round trip times being largely due to processing delay.

TCP Flow Duration and Size Given that the ideal TCP flow for a control system environment would be as long lived as possible to reduce latency, evidence of similar configuration issues can be seen in Figures 3.20 and 3.21. For example, the same spike at 20 seconds is still present in Dataset A3, and the other substations in Datasets C and D still are plagued with numerous short duration flows. Plots for Dataset B have been omitted because of the short duration preventing any useful observations. Again all distributions were fit to Pareto and log-normal distributions using MLE and illustrated on Q-Q plots to qualitatively test for distribution fitness. Note that extreme outlier quantiles were omitted again to allow for closer examination of the main body, and again the anomalous behavior

prevents the empirical distributions from fitting any of the theoretical ones.

Table 3.8: Round Trip Time Means

Between RTU and	μ_{A3} [ms]	μ_C [ms]	μ_D [ms]
Type A.1a	18.1	-	17.4
Type A.1b	17.4	-	12.5
Type A.2	19.0	20.3	17.2
Type B	1.6	0.4	-
CC	5.4	4.5	7.2

Table 3.9: Round Trip Time Standard Deviations

Between RTU and	σ_{A3} [ms]	σ_C [ms]	σ_D [ms]
Type A.1a	7.35	7.4	8.3
Type A.1b	7.74	8.12	5.8
Type A.2	7.26	7.13	7.6
Type B	2.86	2.86	-
CC	2.8	2.8	1.6

Round Trip Times Due to the shorter capture times, no TCP SYN handshakes were observed in Dataset B, and only certain device types were observed in Datasets C and D. The remaining results in Table 3.8 and Table 3.9 do however support the previous observation that RTTs are surprisingly large and variable. Furthermore, the fact that the measurements are so similar with device types even across substation networks further suggests that they are primarily dependent on device processing time rather than network architecture.

Table 3.10: Hourly Retransmission Statistics

	μ_{total}	σ_{total}	μ_{ACK}	σ_{ACK}
A3	1.32%	0.33%	24.1%	2.21%
B	0.858%	0.016%	37.5%	0.975%
C	0.298%	0.022%	74.8%	5.25%
D	1.17%	0.027%	28.6%	0.456%

TCP Retransmissions Similar high retransmission rates were seen across all substations with the notable exception where the smallest substation, Dataset C, had only 0.298%

of all packets retransmitted. Although omitted here due to space constraints, retransmissions were again uneven between the master and field devices due to the apparent widely different calculations of the RTO.

Since the same device types are used in all of the substations, they all use the same RTO calculation algorithms resulting in similar RTO distributions in Figure 3.25, with noticeable clusters at 1 second, 3 seconds, and 300ms.

3.6 Discussion

Based on the measurements taken during this research, three major areas of discussion and impact arise.

ICS Network and Device Implementations Throughout the course of this research, one of the recurring observations made was that there were widespread implementation issues causing non-ideal network performance. Although the dataset used in this research is not large enough to conclusively determine how many other ICS networks suffer similar issues, the problems appear in several popular vendors and in multiple layers of the protocol stack suggesting that it is most likely indeed a widespread problem. Examples of the variety of issues observed in this dataset include overwhelmingly large numbers of TCP connections closing ungracefully, broadcast storms occurring despite the use of the spanning tree protocol, and disagreements in the implementation of DNP3 resulting in TCP connections being unnecessarily closed. Other interesting observations revealed surprising amount of jitter in the polling intervals initiated by software running on one of the most popular real-time operating systems in the world.

Since there are clearly efficiency issues in the implementations of the protocol stacks in these devices, it also suggests a strong likelihood of the presence of security vulnerabilities. In fact, a study conducted in 2014 found widespread vulnerabilities in power grid devices [13], and through the course of this research multiple ICS-CERT advisories were also released [36] [3] [37]. The fact that so many devices are still plagued with easily observable

vulnerabilities that have been around for decades highlights how vulnerable these networks are and how much more attention should be paid to them.

Usage of TCP The second main observation this research offered was that due to the steady low bandwidth requirements of embedded ICS devices, the TCP protocol performs several functions that are completely unnecessary, including congestion control. Combined with the fact that storage space and processing power are both very limited on these devices, it could be very desirable to leave these functions out. The official DNP3 specification even recommends that it may be more efficient to use UDP as opposed to TCP when implemented over LANs where packet reordering is not present. However, if reliable transmission is required, then the DNP3 level acknowledgments and retransmissions would have to be used, which have not been as thoroughly studied as TCP retransmission algorithms. Due to these reasons then it may be desirable to make a compromise between efficiency and functionality and use a stripped down version of TCP/IP like uIP developed by Adam Dunkels [38].

Round Trip Times The final and perhaps the most interesting observation is that traditional RTT estimation, and by extension traditional TCP retransmission algorithms, do not perform as expected in ICS environments due to the slow processing of the embedded devices. This resulted in significant redundant retransmissions and wasted bandwidth that could have been saved if a smarter RTO algorithm was used. Furthermore, as a result of the RTTs being so dependent on processing as opposed to propagation and queuing delay, it appears that measuring RTTs reveals information as to the device's processing power and identity. As the Internet of Things continues to grow with ubiquitous embedded devices, this observation could be crucial to reducing redundant retransmissions and possibly even be leveraged for device fingerprinting and intrusion detection.

Generality Being a case study, by definition, this research only focused on the networks of a single ICS. However, several of the most interesting observations have nothing to do with the specifics of power substations and hold true for any network of embedded

devices, including other ICSs. Embedded devices by their nature are designed to perform very specific tasks, which results in limited processing power and memory and equivalently, limited capability for generating large amounts of traffic. Furthermore, when they are running application layer protocols designed to regularly poll for measurements, it virtually guarantees a constant bandwidth usage and packet size. When these devices, which are still often designed to be compatible slow legacy serial links, are networked together with common IT technology capable of speeds of 100Mbps and greater, traditional notions of round trip times and critical concerns for congestion control no longer hold.

3.7 Conclusions

In this chapter we present the most detailed characterization of power substation network traffic to date and examine how well the behavior of the target networks align with common assumptions about ICS and SCADA systems. We found that while most assumptions held true, there was also a surprising amount of unexpected behavior including slow and variable round trip times dominated by processing time, relatively high retransmission rates, and polling intervals with large jitter. Furthermore, evidence suggested that most of the various functions that TCP provides, including congestion control, are largely irrelevant in the ICS environment. These insights and observations are crucial to creating more accurate ICS network simulations and inspiring several areas of new research. Finally, it was observed that devices were heavily clustered based on their RTTs, no matter how far away they were from the master device. This suggests that the processing power on these resource-starved embedded devices could be leveraged for identifying information that is necessary for fingerprinting. This idea is studied further in Chapter 4 and expanded further in Chapter 5.

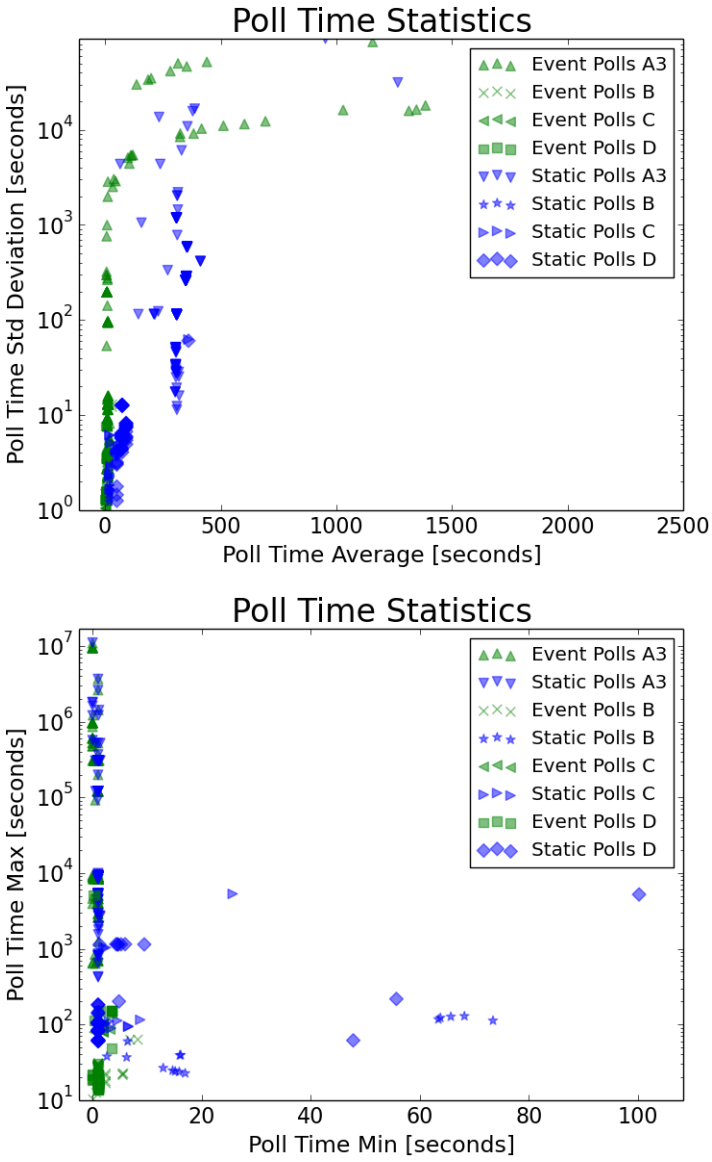


Figure 3.16: Polling Intervals

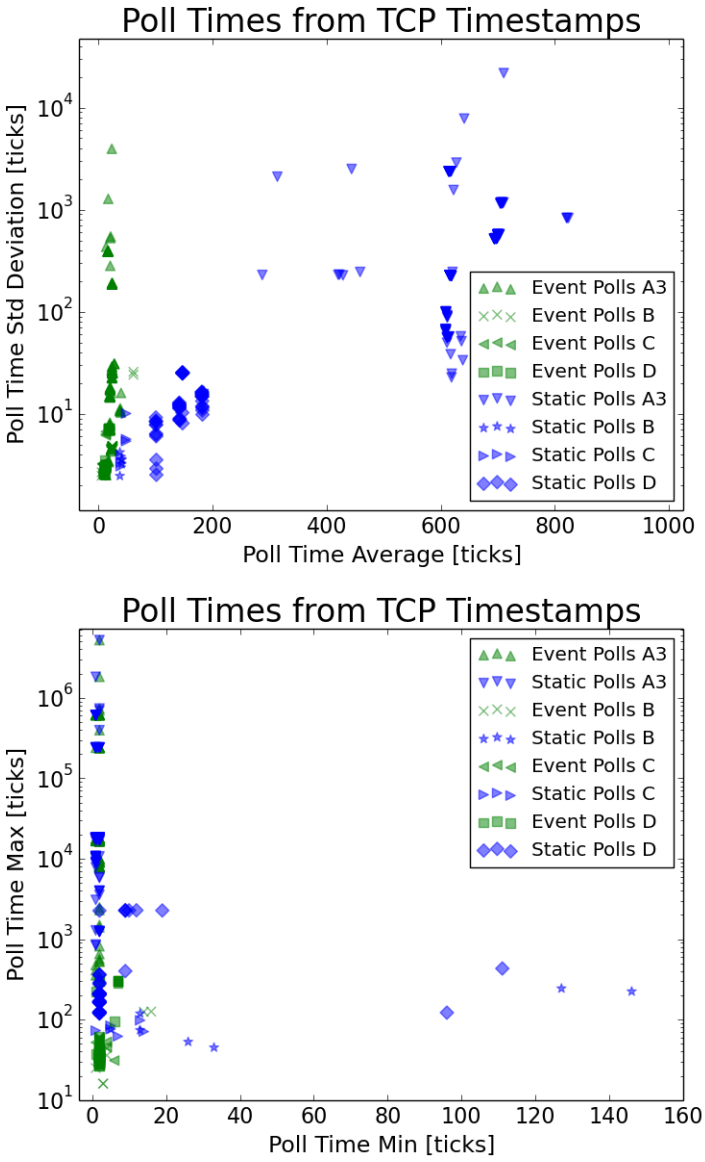


Figure 3.17: Polling Intervals from TCP Timestamps

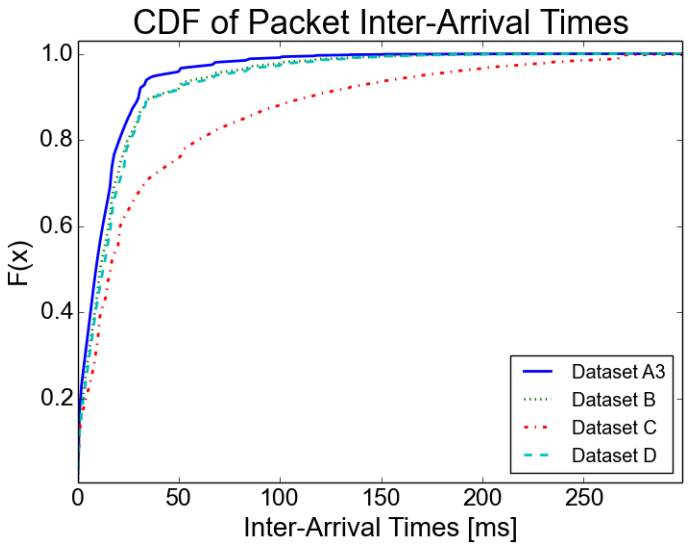


Figure 3.18: CDF of Packet Inter-Arrival Times

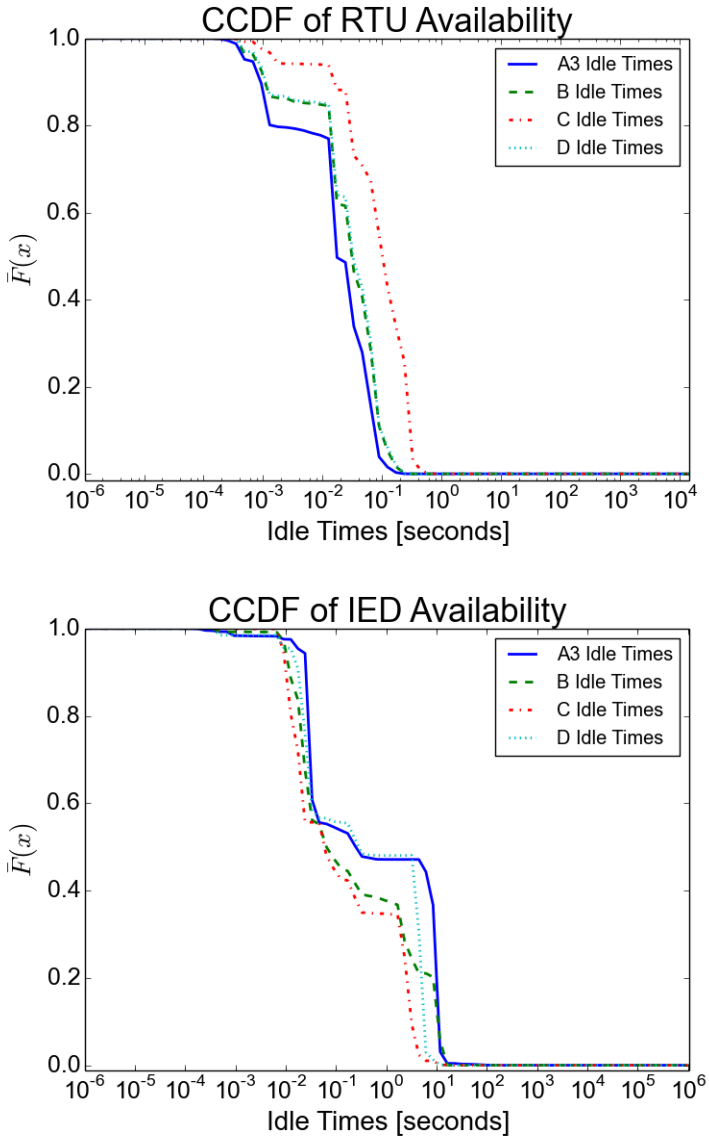


Figure 3.19: CCDFs of RTU and IED Idle Times

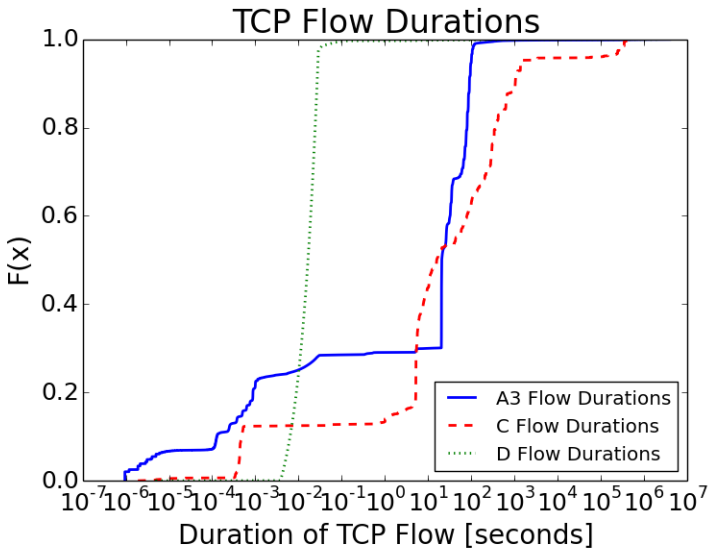


Figure 3.20: TCP Flow Durations

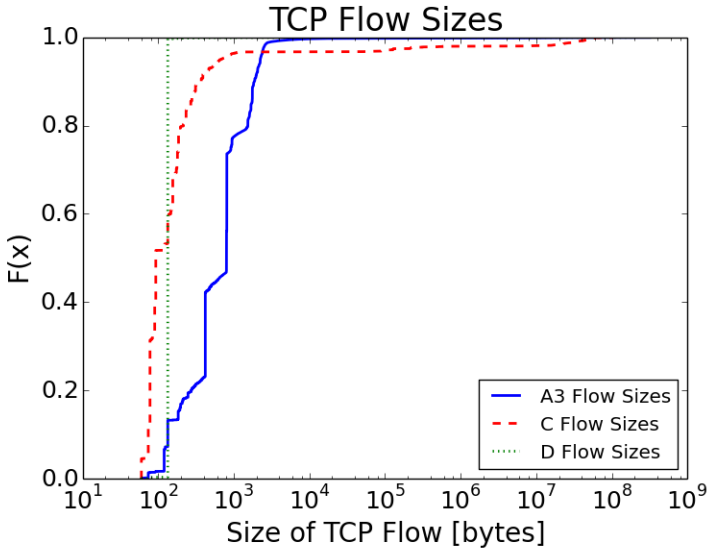


Figure 3.21: TCP Flow Sizes

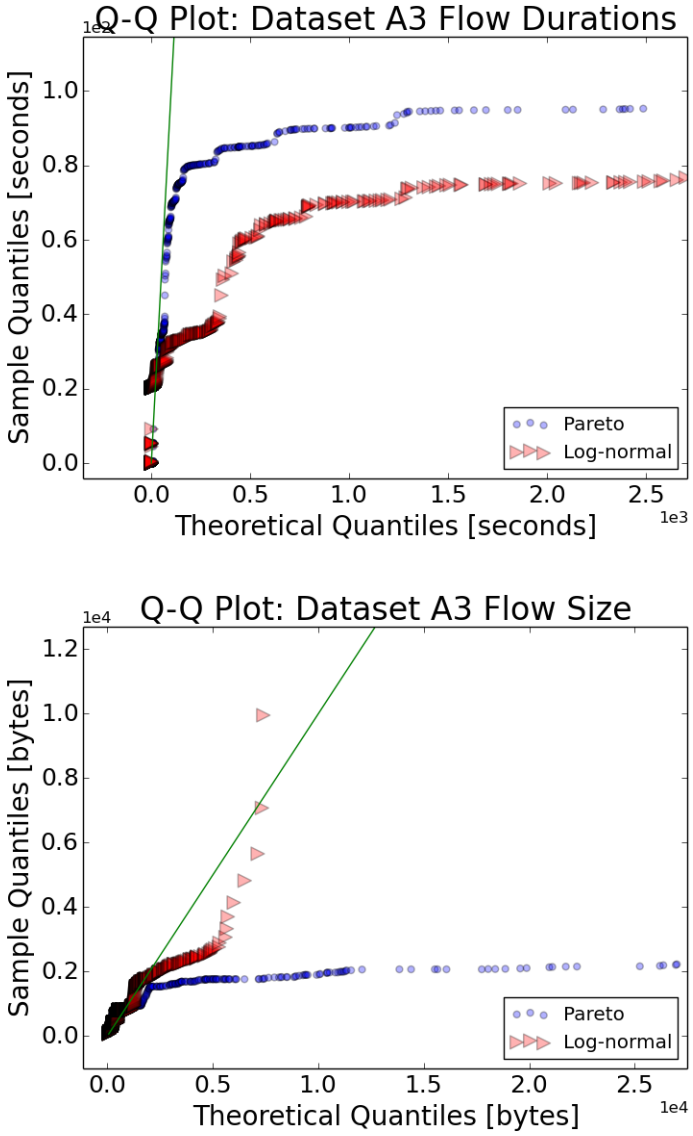


Figure 3.22: A3 Q-Q Plots

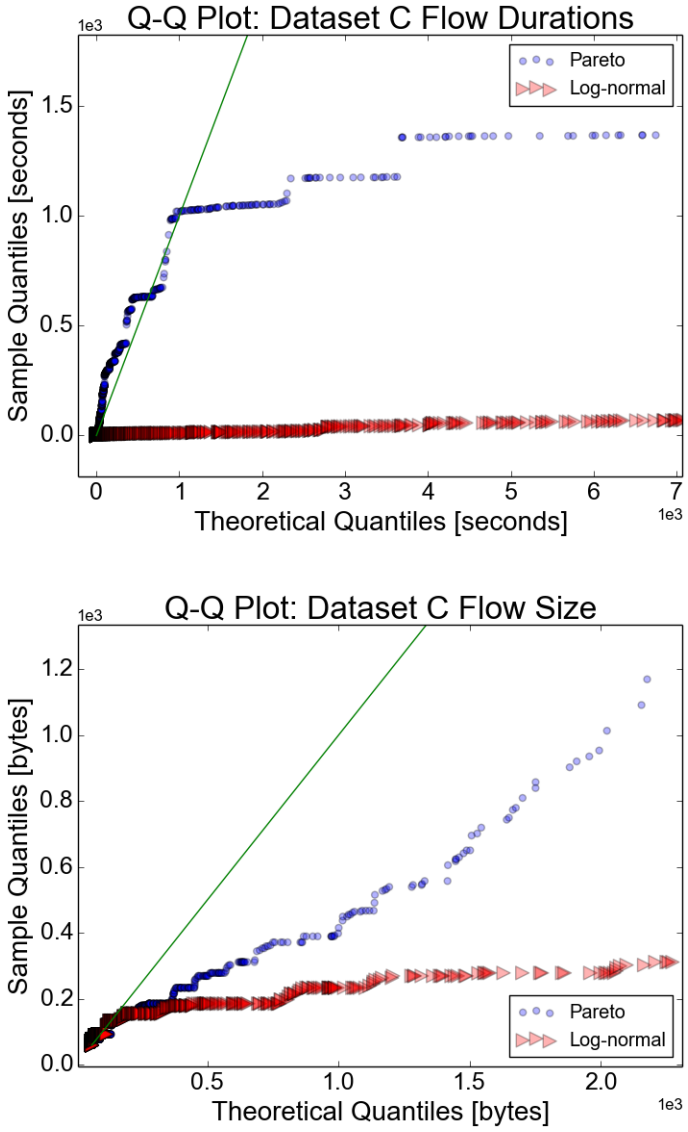


Figure 3.23: C Q-Q Plots

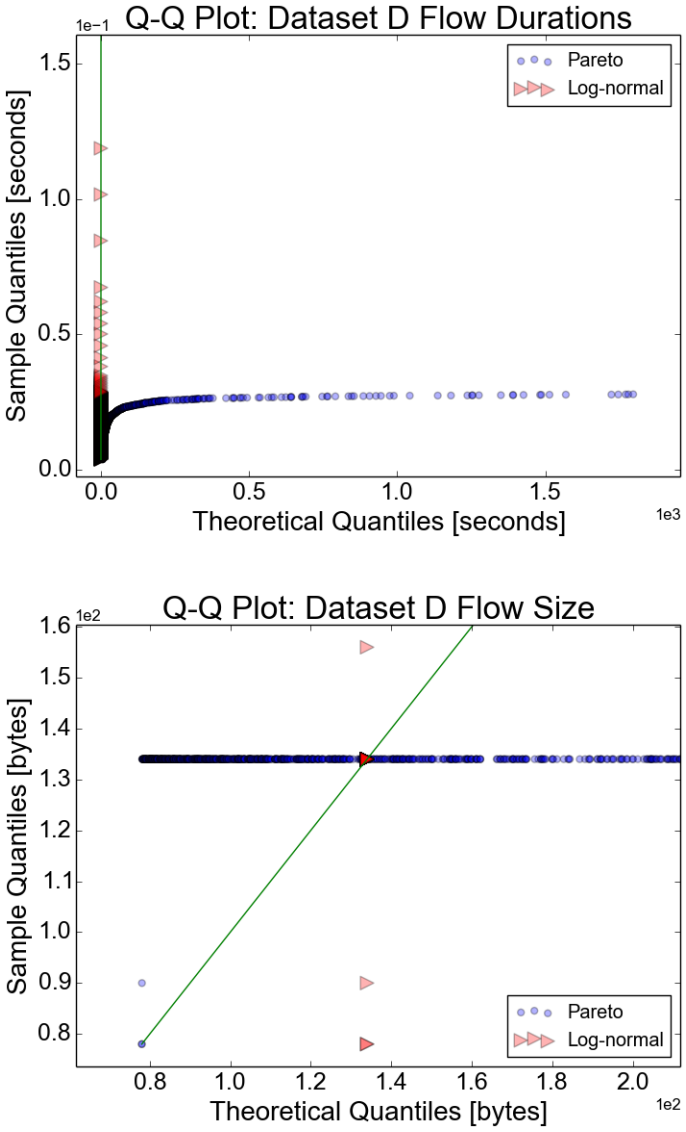


Figure 3.24: D Q-Q Plots

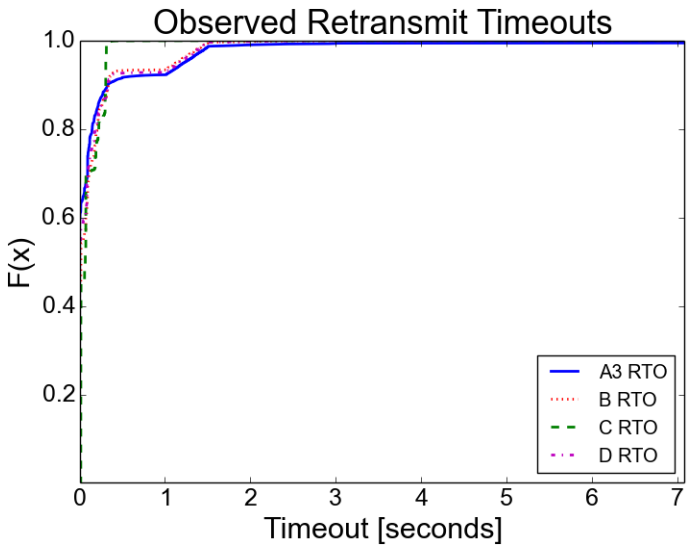


Figure 3.25: Retransmit Timeout Distributions

CHAPTER 4

CYBER-PHYSICAL SYSTEM FINGERPRINTING

4.1 Introduction

Fingerprinting devices on a target network, whether it is based on their software or hardware, can provide network administrators with mechanisms for intrusion detection or enable adversaries to conduct surveillance in preparation for a more sophisticated attack. In the context of industrial control systems (ICS), where a cyber-based compromise can lead to physical harm to both man and machine, these mechanisms become even more important. An attacker intruding on an ICS network can theoretically inject false data or commands and drive the system into an unsafe state. Example consequences of such an intrusion can range from widespread blackouts in the power grid [1] to environmental disasters caused by tampering with systems carrying water, sewage [2], oil, or natural gas. These false data and command injections could be thwarted using strong cryptographic protocols that provide integrity and authentication guarantees, but in ICS networks it is often infeasible to upgrade legacy equipment to provide them due to lack of processing power, devices being in remote locations, and the critical nature of the systems that must be online at all times. In fact, some vendors do not even support the functionality of upgrading devices to install critical patches. When our previous research found vulnerabilities in several power system devices and they were reported to ICS-CERT, the resulting official advisory for one of the products stated that "There is no method to update [GE] Hydran M2 devices released prior to October 2014 [39]." Since adding cryptography to resource limited devices and keeping them patched is infeasible and sometimes just impossible, alternative methods such as fingerprinting must be used to provide security and intrusion detection.

While device fingerprinting is a well-studied topic with several solutions already pro-

posed, none of them until recent work [4] were properly suited for the ICS environment. Active fingerprinting techniques can achieve high accuracy detection of operating systems and server versions, but require probing the network with specially crafted packets. This solution is undesirable in an ICS environment where devices are performing time-critical functions and administrators would rather not risk even the small chance of a port scan crashing the legacy devices and resulting in critical system downtime that includes loss of revenue and potentially life-threatening situations for affected customers such as hospitals. Therefore passive techniques are more suited, but they usually provide limited useful information or require special equipment or TCP options enabled.

This chapter presents some of the defining characteristics of ICS networks and discusses how to use them to develop two fingerprinting approaches that perform uniquely well in the ICS environment, where the two primary functions are *data acquisition* and *control* and are carried out over supervisory control and data acquisition (SCADA) protocols such as DNP3, Modbus, and IEC 61850 GOOSE. Our first fingerprinting approach uses the control aspects of ICS environments to extend previous work on generating signatures from the physical operations being taken by the physical devices on the network. Even though two relays or valves from different vendors may have similar ratings, there will always be physical variations in their construction resulting in fundamental differences in their operation times. These differences are then used to identify device types or spoofed command responses, which we call physical fingerprinting. The basics of this idea was recently proposed in previous work [4], but was limited in scope and accuracy by several assumptions. In this chapter we significantly improve on the work by removing these assumptions and drastically increase the accuracy and scalability. Our second approach takes advantage of the data acquisition functions by using the interaction between the application layer responses and transport layer acknowledgments to obtain measures estimating the speed and workload of a particular intelligent electronic device (IED). Due to the unique properties of ICS networks, the distributions of these measurements are constant within device types

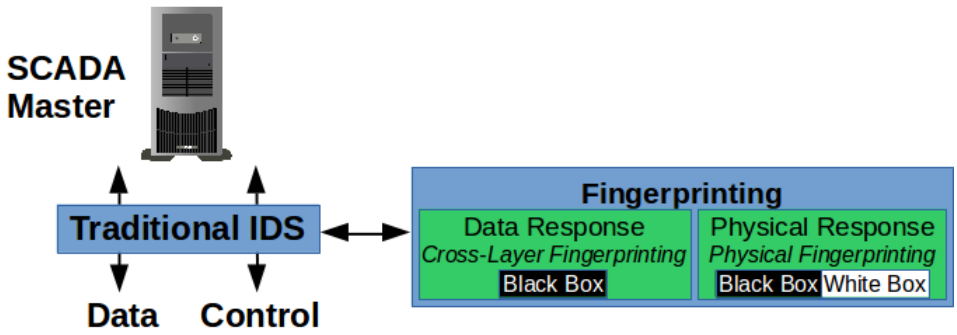


Figure 4.1: The two novel fingerprinting methods can work together to augment traditional intrusion detection

and software configurations allowing network administrators to passively detect changes in the configuration or spoofed communication. Throughout this work we refer to this technique as cross-layer fingerprinting. When used together as illustrated in Figure 4.1, these two methods can achieve device fingerprinting from software, hardware, and physics based perspectives and provide a strong supplement to more traditional intrusion detection systems (IDS) in the ICS environment.

The fingerprint (or signature) of a device can be represented as a probability density function (PDF) of the response times described above. To generate these PDFs, one of three approaches can be used: white box, black box, and gray box modeling. In a white box approach, a dynamic model of the device is constructed from first principles and model parameters identified from CAD drawings, source code, physical measurements, etc. without ever seeing any true samples from the system. The simulated behavior is then used to create a PDF by varying model parameters using an uncertainty distribution. In a black box approach, the PDF is constructed strictly from experimental data without any dynamic modeling, requiring a significant amount of experimental measurements but little knowledge of the underlying system. Finally, in a gray box approach, a dynamic model is first constructed and the resulting PDF is then refined based on experimental measurements. White box modeling is best suited for when a system's internal details are accessible but access to experimental measurements is restricted. Black box modeling performs best when

experimental measurements are easily available and especially when the system is proprietary or too complex to model. Finally, gray box approaches are most advantageous when the basic characteristics of a software or hardware design are known, but there is some uncertainty in model structure or parameters that can only be dealt with through experimental observations [40].

Due to the abundance of measurements in the available dataset and lack of proprietary source code, the data acquisition fingerprinting method proposed here, called cross-layer fingerprinting, focuses on a black box modeling approach. In the case of the physical fingerprinting technique, there are some devices where the operations occur so rarely that collecting enough real samples to generate an accurate fingerprint through black box modeling is completely infeasible. To address this, previous work on physical fingerprinting laid out the process for generating white box model fingerprints for the physical devices, giving administrators usable fingerprints without the need for real samples. In this extension of the physical fingerprinting, only black box models are used to prove the feasibility on other types of physical devices.

The major contributions of this chapter include:

- Novel device fingerprinting approaches that take advantage of the unique characteristics of ICS devices
- Extended physical device fingerprinting with improved accuracy and scalability
- Performance analysis using both real world data from a power substation and controlled lab tests

The remainder of this chapter is organized as follows. First, the assumptions and threat model addressed by this work are presented in Section 4.2. Section 4.3 focuses on the physical fingerprinting techniques, including a review of the theory and previous results, as well as the proposed extensions. Next, the cross-layer response time methods for fingerprinting embedded devices is discussed in Section 4.4. Finally, the performance and limitations of

the techniques are discussed in Section 4.5, and the results and next steps in the work are summarized in Section 4.6.

4.2 Threat Model, Assumptions, and Goals

One of the unique challenges for ICS network security is the vast attack surface available due to the distributed nature of many ICS networks. For example, the electric utility from which experimental data was gathered for this research covers an area of *2800 square miles with 35 substations*, where each substation serves as a point of entry to the network. With such a large area to cover, physical security is extremely difficult to achieve [41]. Therefore, we consider two different attacker models: 1) an outsider who is unable to gain physical access but has compromised a low powered node in the network with malware, and 2) an outsider who is feasibly able to gain physical access to the target network and use her own portable machine with standard laptop computing power. The first attacker model was chosen due to how vulnerable these devices are (as evidenced by the 30 year old TCP vulnerabilities found widespread in the power grid [13]) and because it was the method used on the most well known ICS attack to date, Stuxnet [42]. The second attacker model is realistic in the scenario of a widely distributed control system where physical security is difficult to achieve.

The adversary's goal in this chapter is to inject false data responses, false command responses, or both. As already discussed in the in Chapters 1 and 2, false data and command injections such as these can have disastrous effects on the power grid. With this in mind, the goal of this research is to develop accurate fingerprinting methods to identify what *type* of device these responses are originating from as opposed to unique devices. Such methods could be crucial to distinguishing between responses originating from a legitimate IED, an adversary with a laptop who has gained access to the network, or a comprised IED posing as a different device on the network.

For a formal definition, assume the global set of all ICS devices G consists of products

$D_{j,k}$, where j identifies the vendor and k signifies the model for each vendor's product. Given a sequence of observations O_i every device i on the network, the goal of the fingerprinting methods will be to identify which subset of G, specifically which $D_{j,k}$, those observations belong to.

4.3 Physical Fingerprinting

Two of the properties that differentiate ICS networks from more traditional networks are their primary functions of data acquisition through regular polling for measurements and control commands. These properties hold true for all of the most critical ICS networks regardless of the underlying physical process, including the distribution of power, water, oil, and natural gas. The methods proposed below take advantage of these unique properties and are explained first using the power grid as a specific example. The first proposed fingerprinting approach addresses the control half of SCADA systems by extending previous techniques for fingerprinting physical devices based on their unique physical properties [4] [43]. Based on the differences in methods used to collect and generate the physical fingerprints, we label the previous work as Event-driven Physical Fingerprinting and the proposed work as Behavior-driven Physical Fingerprinting.

4.3.1 Review: Event-driven Physical Fingerprinting

In order to understand the proposed Behavior-driven Physical Fingerprinting (BPF), it is first important to provide a detailed explanation of the previous Event-driven Physical Fingerprinting(EPF) and compare them.

Review: EPF Theory

The foundational theory for both EPF and BPF are the same: differences in the mechanical design of physical devices dictate how fast the device operates. However, there are important differences in the assumptions made to apply this theory. In the previous EPF research

on physical fingerprinting, operations are simply assumed to be binary. In other words, operation time samples were single time measurements taken for the whole operation, ignoring the *behavior* of the transition between operational states and instead focusing on the final *event*. Unfortunately, there are several disadvantages to making this simplifying assumption. First, the concept of an operation being "complete" is very clear in the context of latching relays discussed in the previous work, but it becomes much more ambiguous for non-binary operations. If a command is sent to accelerate a centrifuge to a certain speed, some of the ambiguities that arise include questions about how close to the target speed is close enough and whether or not the operation is "complete" while the speed is still settling to a steady state. In the new BPF research here, we address these questions by generating physical fingerprints from the behavior of the transition between states, offering much richer information for classification and allowing for more general applications.

The previous EPF work also described a physical fingerprint as an estimate of the PDF of the operation times. Formally, we defined a physical fingerprint according to Equation 4.1 [43], where M is a set of operation time measurements from a specific device, B defines the number of bins in the histogram (and equivalently the number of features in the signature vector), and H signifies the heuristic threshold chosen to be an estimate of the maximum value an operation should ever take. The range of possible values is divided by thresholds t_i where $t_i = i\frac{H}{B-1}$, and each element of the signature vector is given by s_j.

$$s_j = \begin{cases} |\{m : t_{j-1} \leq m < t_j, m \in M\}| & 0 < j < B \\ |\{m : m > H, m \in M\}| & j = B \end{cases} \tag{4.1}$$

In the new BPF fingerprinting proposed here, we consider more general types of fingerprint signatures to be able to capture information about the transition.

To illustrate the main theory behind both EPF and BPF, we review the simple case of a latching relay that was the focus of previous research. Relays are commonly used in ICS networks for controlling and switching higher power circuits with low power control

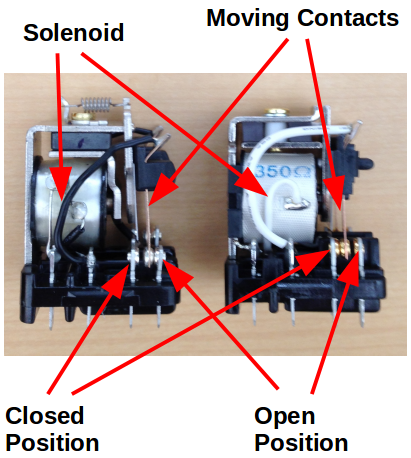


Figure 4.2: Diagram of two different latching relays [4]

signals, and operate by using an electromagnet to move a contact that either makes or breaks an electrical circuit. The relays used in previous research as well as this new research are illustrated in Figure 4.2. From the figure, it is clear that there are several opportunities for differences in design to lead to different operating characteristics, including the length of the contact arm, the distance the arm travels, the choice of spring vs permanent magnet, and the strength of the electromagnet. Going even deeper, the magnetic force produced while energizing the electromagnetic core in a latching relay is directly proportional to current though the windings (affected by the resistance of the windings), number of windings, the cross sectional area of the core, and the material chosen for the core. Since this magnetic force is the primary factor that determines how fast the relays operate, any variation in one of the variables will contribute to differences in operation times.

Review: Measurement Method

To measure the operation time, previous research used the setup illustrated in Figure 4.3, where the intelligent electronic device (IED) provided sequence of event recording (SER) that timestamped when changes were detected at the inputs. The operation time t_o was then measured by the equation $t_{op} = t_2 - t_1$ where t_2 was the timestamp from the IED and t_1 was the time that the network tap observed the command being sent. Note that this

setup requires the network tap and IED to keep tightly synchronized clocks for accurate measurements, noise is introduced by random network and processing delays, and the SER timestamps only have millisecond precision. These assumptions significantly limit the accuracy and usefulness of the fingerprinting techniques and are addressed in the extensions proposed here. In short, the BPF methods remove any effects from the network by measuring operation times locally on the end devices themselves. This setup is explained in more detail in Section 4.3.2.

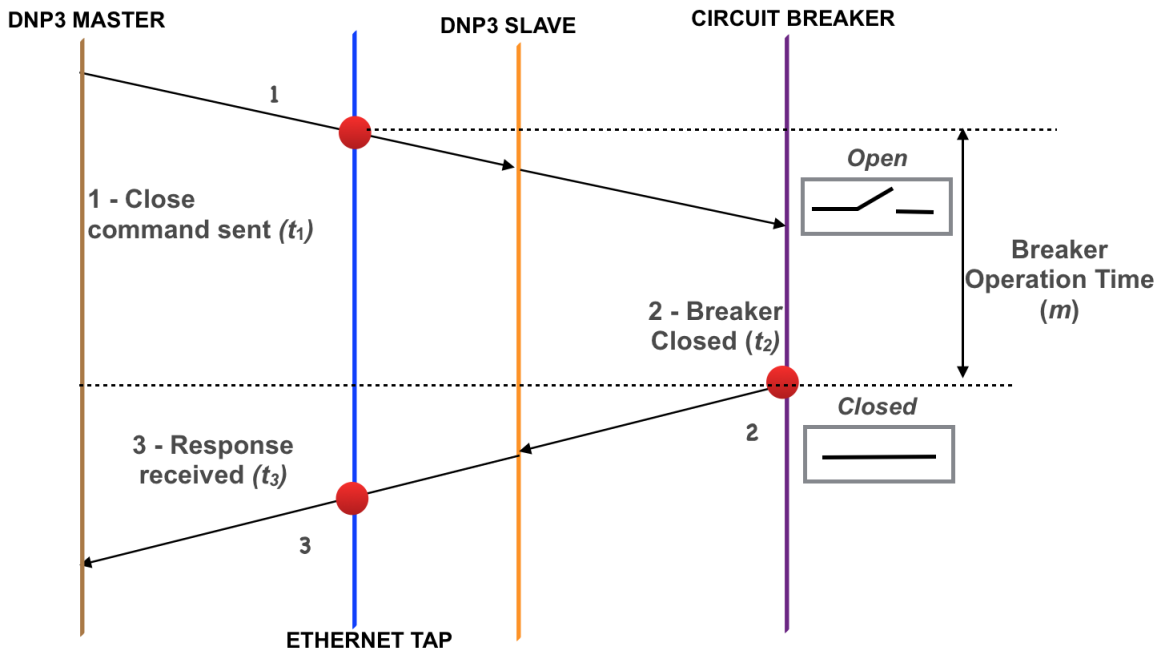


Figure 4.3: Timing diagram to calculate Operation times [4]

Review: Results

Using the EPF methods, the previous work found distributions of close operation times for relays from two different vendors to have noticeable differences, illustrated in Figure 4.4. The operation times ranged from 16ms to 38ms for Vendor 1 and 14ms to 33ms for Vendor 2, resulting in useful information for classifying between the two device types. Due to the inaccuracies in measurement, distributions for "open" operation times were found to be too similar to be differentiable.

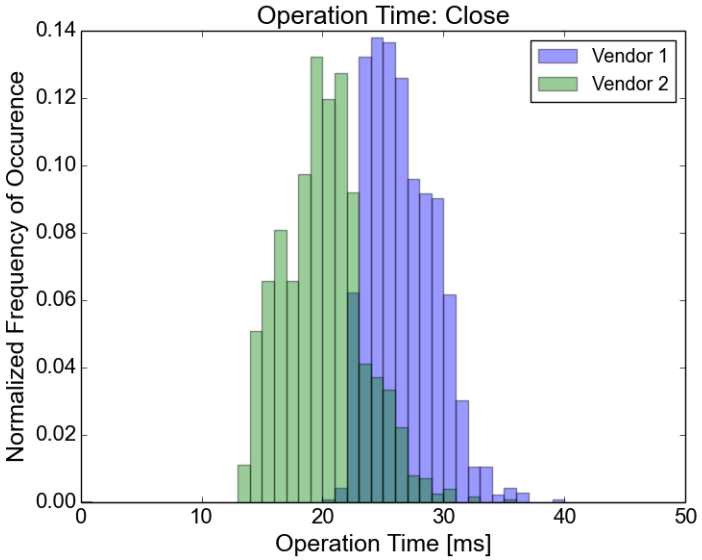


Figure 4.4: Distribution of Close Operation times based on SER Responses [4]

To test classification performance, previous research applied a naïve Bayes classifier with varying levels of sample requirements. At the lower end, attempting to classify a device after only observing one operation sample resulted in an accuracy of around 76%. At the upper end, when the classifier was allowed to wait for 32 samples before attempting classification, accuracies of around 92% were reached. Note that this illustrates another important limitation of the previous EPF methods: acceptable classification performance was only achieved with relatively large numbers of operation samples. Given that the frequency of operation for breakers in the power grid is on the order of days or weeks, waiting for 32 samples of breaker operations would take an infeasible amount of time.

Review: Synthetic Fingerprint Generation

While the results obtained in the previous research for physical fingerprinting were promising, the fingerprints were generated using black box methods that assume some access to the target devices. Alternatively, physical fingerprinting techniques may leverage a white box, black box, or gray box modeling approach since the mechanical composition of a device can usually be obtained from manual inspection, available drawings/pictures, or manufacturer's specifications. The ability to construct white box model fingerprints for physical fingerprinting is crucial due to the rare operation of some devices, and the prohibitive cost of performing black box modeling on all of the available devices on the market. To illustrate this technique, previous research described construction of the same fingerprint for the latch relay mechanism discussed in Section 4.3.1 using white box modeling only and then validated it against the black box model results obtained for the device in Section 4.3.1.

To test how well this white box modeled "synthetic signature" could be used in fingerprinting, the same machine learning techniques were applied as before, but trained from the simulated distribution for one device and experimental measurements from the other device. An artificial neural network was trained using the same number of samples for each device, and then performance was tested using an equal number experimental measurements for each device. With classification accuracy leveling off around 80%, the white box model expectedly did not perform quite as well as the black box method based on true measurements due to the various simplifications and estimations made during the modeling process. However, the results were still very promising for this new class of fingerprinting. Furthermore, in a real world scenario the white box model approach would be limited to scenarios where there is not enough experimental data or the integrity of the experimental data is in question. The white box approach can then be combined with the black box approach to enable gray box modeling where appropriate to achieve higher accuracy. While there are a variety of techniques to approach this problem, Bayesian learning being one, intuitively it is similar to simply replacing synthetic samples in the white box distribution

with real samples over time as they become available.

The BPF methods proposed here do not go through the whole white box modeling process as the previous research did, but for every new device type studied, we offer brief discussion on how it could be accomplished. Note that since the BPF method uses more detailed information about the behavior of an operation compared to the EPF methods, the corresponding white box models also require more detailed information.

4.3.2 Behavior-driven Physical Fingerprinting

The previous work on EPF methods only focused on relays and breakers, which while important, only represent a small and simple subset of the variety of actuators in ICS settings. Furthermore, the techniques presented required very specific conditions where devices' clocks had to stay perfectly synchronized for accurate operation times to be measured. In this section described the Behavior-driven Physical Fingerprinting (BPF) we address both of these limitations by porting the techniques to PLCs, which allows for a more general variety of actuators and more stable timing measurements regardless of the network architecture.

BFP Methods

The experimental setup for the previous work on physical fingerprinting, illustrated in Figure 4.3, measured operation times by taking the difference between the system time that the sniffer observed a command packet traverse the network and the event timestamp recorded in the slave device's response. The sniffer's system clock must stay tightly synchronized with the end device that is recording event timestamps or else the clocks will drift and the measurements will become useless. Additionally, with this measurement setup the further away the network tap is from the slave device, the more "signal noise" is introduced by network and switching delays. Therefore, accurate measurements using the previous methods required networks taps being placed as close as possible to the devices operating the

actuators, which would inhibit the feasibility of a widespread real-world deployment.

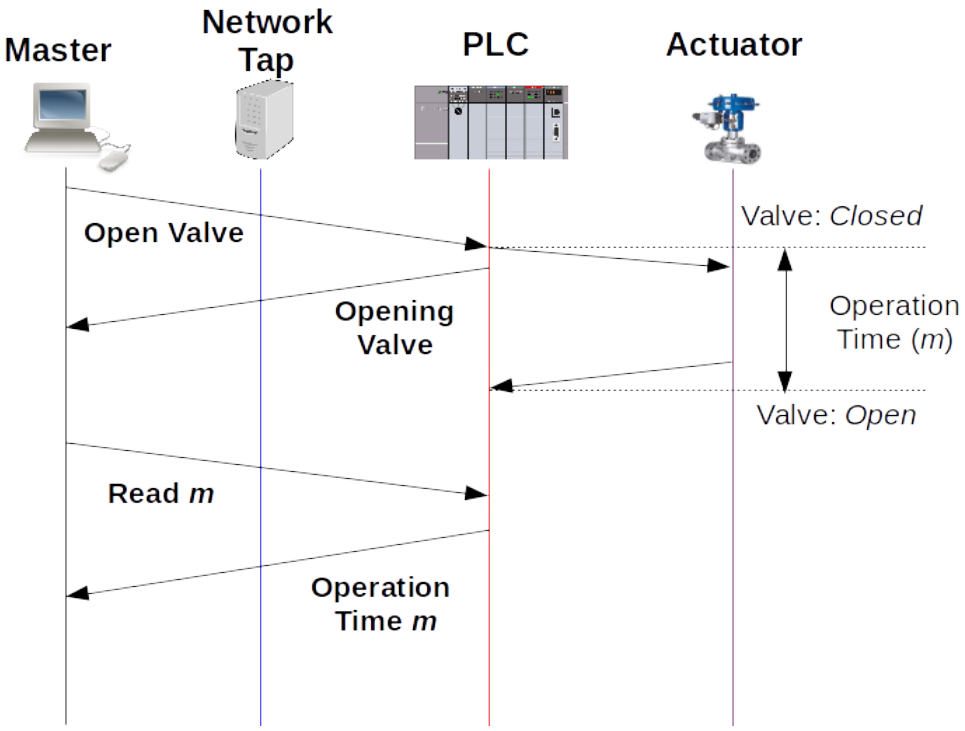


Figure 4.5: BFP Method of Measuring Operation Times

The BPF method improves on the previous EPF method of measuring operation times by using PLCs instead of smart relays, and by adding a minimal amount of extra lines of programming in the PLCs to enable the PLCs themselves to measure the operation time of the actuators they are connected to. The master device in the control system can then poll the PLCs for the operation time as it would for any other measurement of the underlying process. This new experimental setup, illustrated in Figure 4.5, removes any chance of clock drift or network delays affecting the operation time measurements, making the technique much more feasible for widespread deployment. Furthermore, the increased precision in operation time measurements and the richer source of information provided by examining the transitional behavior of the operations both significantly improve classification accuracy for similarly rated devices.

To compare the EPF and BPF methods, we now discuss the theory, measurement methods, and classification results for a variety of devices including latching relays, valves, AC

induction motors, and diaphragm pumps.

Relays

Theory. To illustrate the BPF method, it was first applied to the exact same physical relays from the previous work in Section 4.3.1. The same basic theory from Section 4.3.1 about the latching relays still applies here. Differences in design choices about the various components of the electromagnet, the moving contact arms, and the holding mechanism (spring vs magnet, e.g.) all work together to determine the operation speed.

Method. To measure the relay operation times, a PLC was programmed to measure the time between when it energizes an output to operate a relay, and when its inputs detect that the relay circuit has been closed. Given the extremely fast operation time and the nature of electromechanical relays, it is infeasible to measure transitional behavior for relays. Therefore, the fingerprint signatures for the new BPF methods simplify down into essentially the same signatures as the EPF method, but using the more precise operation measurements taken by the PLC.

Results. Recall from Figure 4.4 that the distributions of close operation times had significant overlap, requiring multiple samples to be taken before an accurate classification could be made, and still only having accuracy levels around 92%. With the more precise measurement supplied by the BPF method, there is now zero overlap in the distributions in Figure 4.6. This means that given a single relay operation time sample, the new BPF methods can immediately classify the relays with perfect 100% accuracy, making a dramatic improvement over the EPF results.

Valves

Theory. Now we consider applying the BPF technique to valves, which represent another large class of ICS actuators. Valves control the flow of liquids and gases in applications ranging from water treatment, oil & gas, chemical manufacturing, and steam power plants.

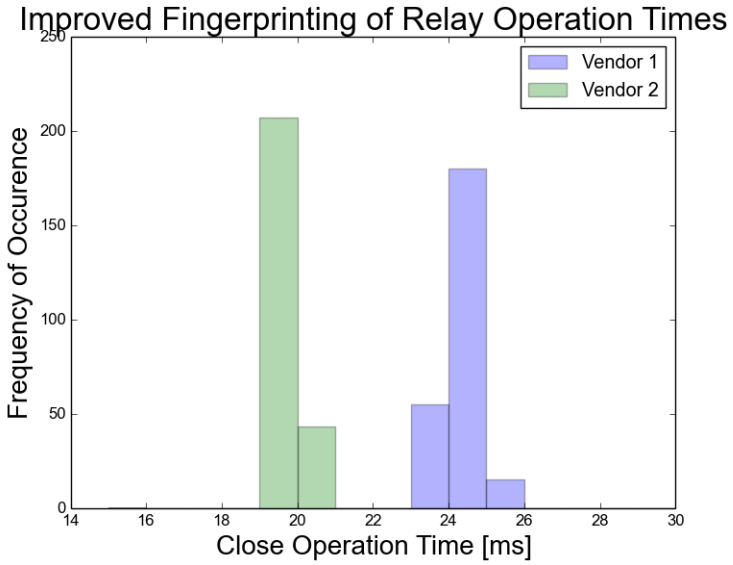


Figure 4.6: Relay Close Operation Times with More Accurate Measurements

They are typically designed to either operate in discrete states (e.g., only fully open or closed) or a continuous range holding a certain position between open and closed to enable precise control of the fluid flow rate and pressure. To make the most direct comparison to the previous work on relays, we first consider discrete valves designed to change between states of being completely open and completely closed. The specific type of valves used here were 90° ball valves with electric motor actuators. As can be seen from the generic diagram in Figure 4.7, these kinds of valves operate by rotating a ball with a cylindrical hole drilled through it. In the "open" state, the hole is directly lined up with the input and output ports, theoretically providing no obstruction to flow. In the "closed" state, the ball is turned so that no part of the hole is exposed to the input and output ports, completely blocking any flow. In "modulating" style valves, the ball can be turned to any position in between these two states to precisely control flow and pressure. The specific models of valves used in this research are illustrated in Figure 4.8.

According to the documentation, both valves have the same "cycle times" (i.e., operation times) of four seconds. In the simplest form, this cycle time t is a function of the speed of the motor v (in rotations per minute), and the radial distance r it has to rotate the

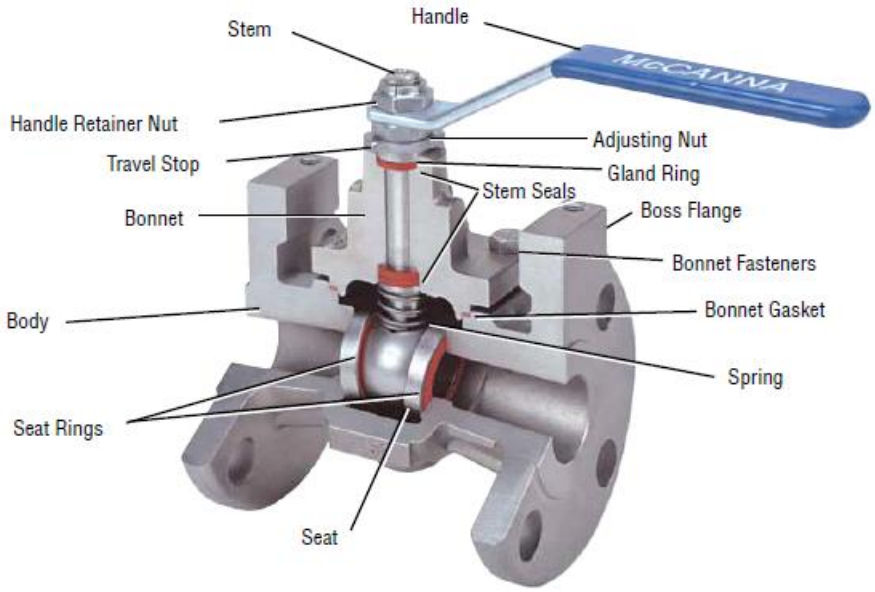


Figure 4.7: Diagram of manual ball valve [44]

ball valve (one quarter rotation), or simply $t = r/v$. Ideally, these valves rotate exactly the same distance using motors that rotate at the exact same speed. However, minor differences in the manufacturing processes for all of the internal physical components contribute to real-world differences. In reality, the radial distance that each valve motor rotates is not exactly $90°$, but instead is determined by the calibration of the limit switches indicating the closed and open positions. Variations in the internal resistance of the actuator motors affect the exact torque and speed that the motors are capable of and tiny differences in the ball, stem, gears, lubrication, gaskets and seals can all contribute to different forces of friction encountered when turning the valve.

Method. To measure valve operation time, PLCs were programmed to measure the time between the output was physically energized, to when the valve limit switches indicated the closed or open position. The valves used in this set of experiments were the two electrically actuated ball valves shown in Figures 4.8. Note that the only difference in specification from the vendor comes from the design of the body being two pieces, or three pieces for easier disassembly and cleaning. Otherwise, the two valves should contain the exact same

(a) Valve with 2-piece body (Valve 1)

(b) Valve with 3-piece body (Valve 2)

Figure 4.8: Ball Valves

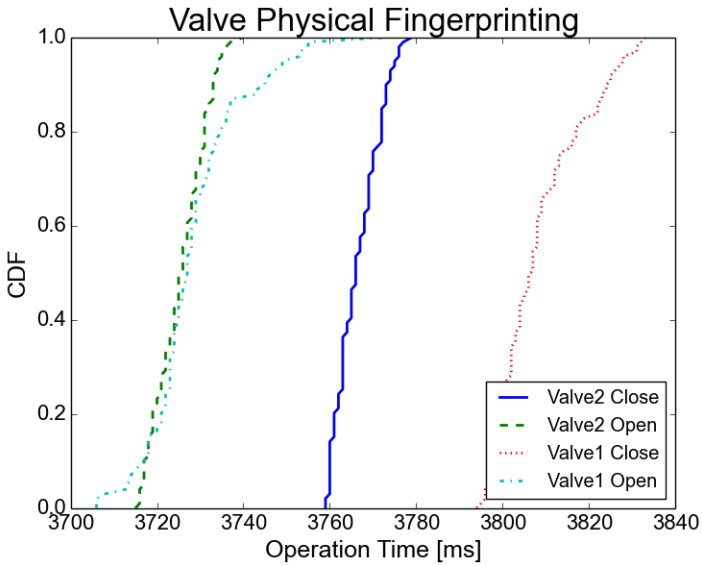


Figure 4.9: Valve Operation Time Distributions

parts including the 110VAC motor actuator and 1/2 inch ball valve.

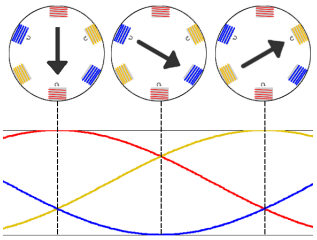
Results. Due to all of the variations in the actual component specifications, the distribution of open and close times for the two valves in Figure 4.9 are easily different. In fact, there is absolutely no overlap between close operation times, meaning a simple threshold check around 3790ms would classify the two valves with perfect accuracy, with no need for any machine learning. Again, it is important to note that this perfect classification accuracy is achieved after a single operation sample, compared to the EPF methods for relays attaining 92% accuracy with 32 samples.

Although beyond the scope of this work, white box models could also be created for valves by expanding on the earlier, oversimplified equation of $t = r/v$. For example, precise measurements could be made to determine the exact rotational distance between the limit switches being "open" and "closed" to fill in for r. Then, the speed v of the valve motor could either be directly measured or estimated from parameters such as internal motor resistance, gear ratios, back EMF (if DC), and AC frequency (if AC motor).

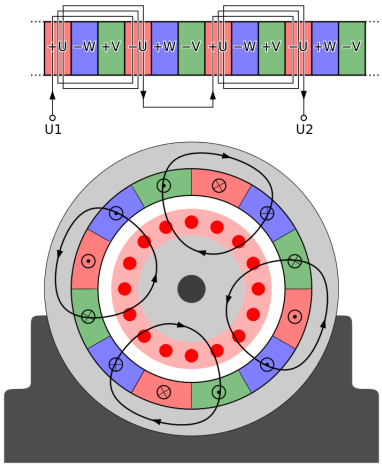
Motors

Theory. The relays, breakers, and valves discussed up to this point all have very clear operational states: open and closed. However, other actuators require more complex descriptions for their states. For example, consider a motor controlling a centrifuge. The motor is not simply on or off, but instead must accelerate to the desired rotational speed, and depending on the tuning of the control loop may hover around the given setpoint. A physical fingerprint for this type of operation must take into account the starting speed, the target speed, and the characteristics of the acceleration to the target speed.

For this set of experiments, 3-phase AC induction motors were chosen to illustrate how to apply Behavior-driven Physical Fingerprinting (BPF) to motors, since they are used in a wide variety of applications including conveyor belts, fans, and centrifugal pumps. These induction motors work by using 3-phase power to create a rotating magnetic field, illustrated in Figure 4.10, that provides the torque to turn the rotor. The synchronous speed n for a given induction motor, which measures how fast the magnetic field is rotating, can be calculated using the formula, $n = 120f/p$, where f is the input frequency and p is the number of magnetic poles in the induction motor [45]. For example, the motors used in this research had 6 poles and were rated for 60Hz input frequency, so they all had a synchronous speed of 1200RPM. To control the actual speed of an induction motor, the input frequency in the previous equation must be changed from 60Hz using another piece of industrial equipment called a variable frequency drive (VFD). VFDs work by taking in fixed



(a) 3-phase waveform and rotating magnetic field



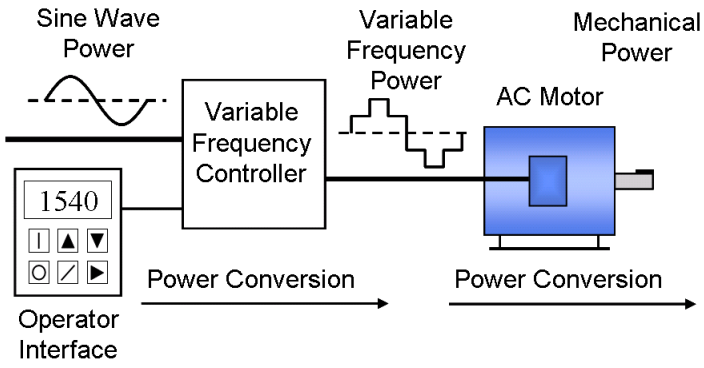
(b) Poles of a 3-phase induction motor

Figure 4.10: 3 Phase Induction Motor Diagrams

frequency (e.g., 60Hz) power and using pulse width modulation to output power at varying frequencies (Figure 4.11a) with higher frequencies causing a faster rotating magnetic field and faster motor speed.

It is important to note, however, that in order to produce the torque necessary to drive any loads, the speed at which an induction motor shaft actually rotates must be less than its synchronous frequency. Therefore, the nameplate RPM rating for induction motors are always given as slightly less than their synchronous speeds, and this difference is called the motor "slip". This slip is directly proportional to rotor resistance and load, so any differences in these variables result in different rotation speeds [46].

Method. To investigate physical fingerprints for motors, three 3-phase AC induction motors were chosen with the same horsepower rating, same synchronous speed, and similar nameplate RPM ratings, illustrated in Figure 4.12. Each motor was then in turn directly controlled by the same VFD in Figure 4.11b, which was monitored and controlled by a PLC. The VFD was calibrated to each motor separately, and the PLC was then programmed to command the VFD to alternate between low and high speeds. The PLC monitored the reported output frequency from the VFD and measured how long it took for the frequency to reach every fifth percentile between the low and high speeds, resulting in 20 samples



(a) VFD converts fixed frequency input power to variable frequency output

(b) Schneider Altivar 12 VFD

Figure 4.11: Variable frequency drives (VFDs)

(a) Motor 1: 1140RPM, 1/4HP

(b) Motor 2: 1140RPM, 1/4HP

(c) Motor 3: 1160RPM, 1/4HP

Figure 4.12: AC motors

along the acceleration ramp.

Results. Due to the extremely similar ratings of the motors and the high quality of the VFD control, the average acceleration ramps in Figure 4.13 look very similar to the naked eye. However, even though the differences are small they are consistent, which means that application of basic machine learning or statistical classification techniques is likely to perform well. To further support the argument made earlier that no sophisticated machine learning techniques are necessary, we simply use the Mahalanobis distance metric to perform the classification. Proposed in 1936, the Mahalanobis distance essentially measures the number of standard deviations away a sample point is from a known distribution in n dimensional space. Formally, the equation is given by $D_M = \sqrt{(x-\mu)\Sigma^{-1}(x-\mu)^T}$, where x is the sample point, μ is the mean of the test distribution, and Σ is the inverse of the co-

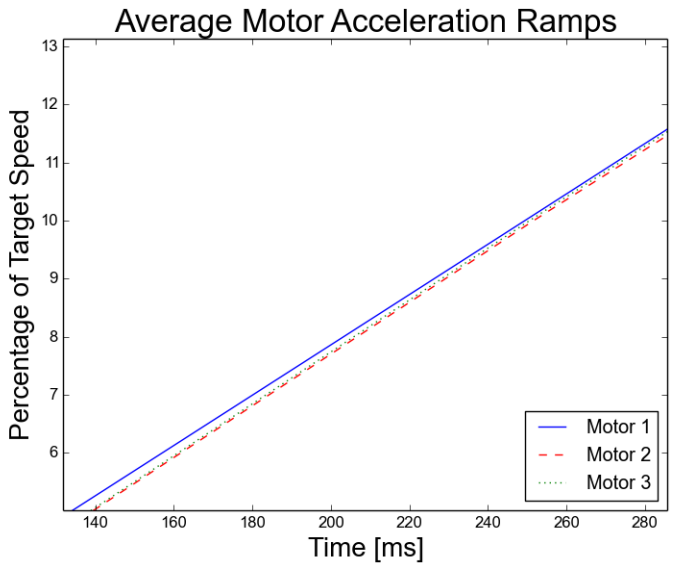


Figure 4.13: Average motor acceleration ramps

variance matrix of the test distribution [47]. In this specific case, each motor's distribution was defined by the 20 dimensional mean of the acceleration ramp and the corresponding covariance matrix.

A total of 100 sample runs for each motor was taken, with half randomly being used for "training" (calculating the mean and covariance matrix), and half being used for test data. For every test sample point, the Mahalanobis distance was first used to determine which motor distribution was closest, and then to determine whether the sample was close enough to that distribution to be positively classified, or to be labeled as "unknown". This confidence threshold for "unknown" classifications was then varied to create an ROC curve in Figure 4.14. Note that after the threshold is large enough, the false positive rate stops increasing and the classification algorithm always picks the motor that is closest and never decides it is unknown, resulting in a maximum true positive rate of about 88% at a false positive rate of 5%. While this performance is acceptable, real world scenarios would provide even more opportunity for higher accuracies where motors are driving different loads.

White box models for motors could theoretically be generated simply from the name-

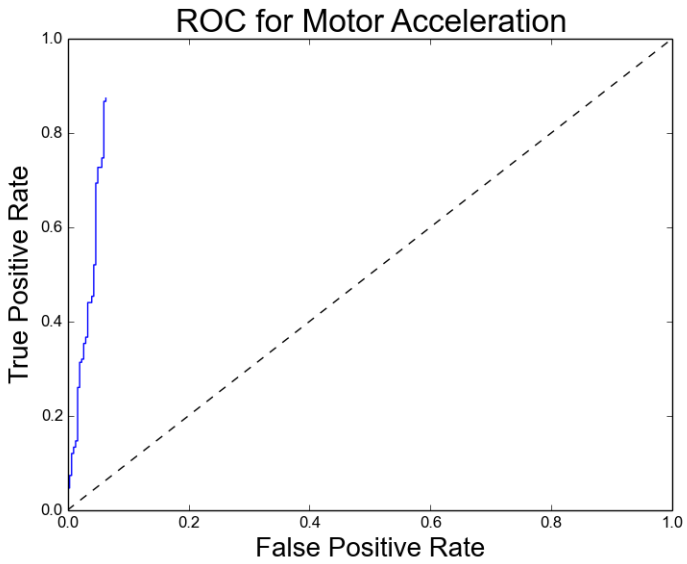


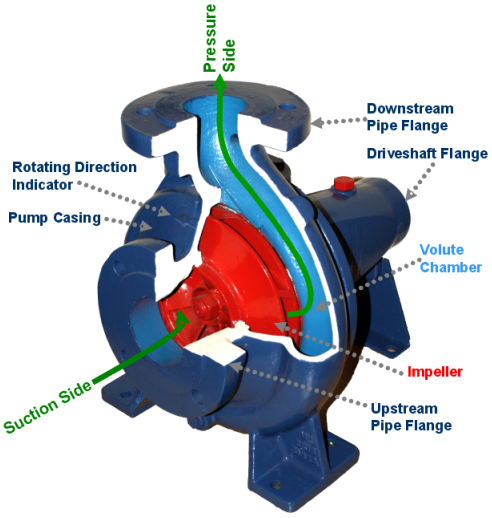
Figure 4.14: ROC Curve for 3-Phase AC induction motors

plate RPM rating and estimate of the load torque, or more detailed measurements of the rotor resistance and motor slip.

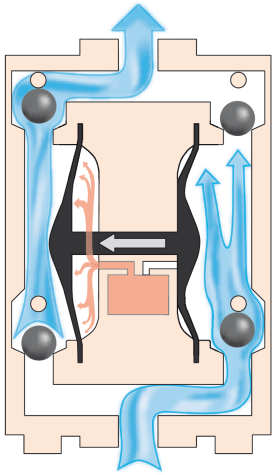
Pumps

Theory. Pumps represent another class of physical devices used in almost every industry. They transport oil & gas, distribute drinking water for a city, and move the water used in steam power plants. One of the many uses for the type of motors examined in the previous section is in centrifugal pumps, which use a rotating impeller to increase velocity and pressure of the input flow of fluid, illustrated in Figure 4.15a. To generate physical fingerprints for centrifugal pumps, the previous methods can simply be used to fingerprint the motors driving the pump. However, other types of pumps are used in industry as well, so more general methods of fingerprinting are required.

To examine general methods for fingerprinting pumps, two diaphragm pumps (Figure 4.16) were chosen with similar ratings. As opposed to centrifugal pumps which translate rotational energy into movement of fluids, diaphragm pumps use alternating linear motion to create the differences of pressure necessary to move fluid, illustrated in Figure 4.15b. Us-



(a) Centrifugal Pump



(b) Diaphragm Pump

Figure 4.15: Example pump diagrams

(a) Pump 1: 12VDC power supply, rated for 3.3 gallons per minute and 35 PSI pressure cutoff

(b) Pump 2: 12VDC power supply, rated for 5.5 gallons per minute and 60 PSI pressure cutoff

Figure 4.16: Small industrial diaphragm pumps

ing an electrical engineering analogy, pumps can be thought of as constant voltage sources, providing a constant pressure differential necessary to move fluid (current) through a pipe (circuit). Depending on the exact speed at which the diaphragm alternates and the volume of fluid displaced by the diaphragm on each cycle, each pump will produce different pressures up to their maximum cutoff limits.

Methods. Since the primary purpose of pumps is to increase flow and pressure of fluids in a system, these were the physical measurements used to fingerprint the pumps. The industrial transmitters used to measure the flow and pressure for these experiments are shown in Figure 4.17, with 2% and 0.5% accuracies respectively.

(a) Flow meter, 2% accuracy with a 15GPM range

(b) Differential pressure sensor, 0.5% accuracy over a 50PSI range

Figure 4.17: Industrial flow meter and pressure sensor

The physical setup for the pumps consisted of the diaphragm pumps lifting water about 3 feet from a reservoir, going through a flow meter, lifting the water another 1 foot to a pressure sensor, and then circulating the water back down to the reservoir. Since this dissertation is for the School of Electrical and Computer Engineering and not mechanical or chemical engineering, it is helpful to make analogies to circuit diagrams we are more familiar with. In this analogy, the typical comparisons are as follows: voltage is equivalent to water pressure, current is the same as flow rate, voltage sources and batteries can be thought of as pumps and elevated containers of water, and resistors can be thought of as constrictions in the piping. With these analogies, the equivalent circuit diagram is represented in Figure 4.18. Note that the "battery" in the circuit represents the vertical section of tubing leading from the horizontal section where the pressure sensor is, back down to the reservoir at standard atmospheric pressure. This means that while the vertical section of the tubing is empty, the pressure sensor is measuring the difference between the initial pressure that the pumps are supplying and the pressure caused by gravity on the left side. After water begins flowing through the entire circuit, gravity on the right side will begin to contribute pressure, resulting in a sudden drop in pressure at the pressure sensor.

To measure the pressure and flow behavior for each pump, a PLC was programmed to conduct 200 cycles of turning on the pumps, sampling the pressure and flow every 10ms for 500ms total, then turning off the pumps letting the system drain.

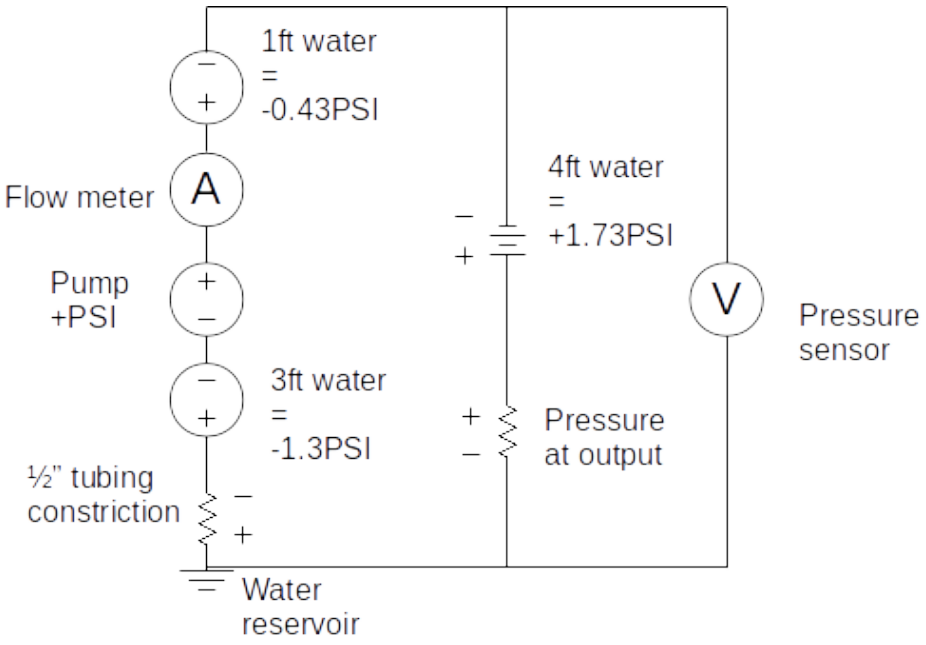


Figure 4.18: Pump "circuit"

Results. The resulting flow and pressure ramps were then averaged to create Figures 4.19 and 4.20. Note that the flow rates created by the pumps increase at obviously different rates, again suggesting any standard classification algorithm would perform well. The same Mahalanobis distance classifier as in the previous section was applied, and the confidence threshold was varied resulting in true positive rates of 94% and zero false positives.

Looking at the pressure behavior in Figure 4.20 again shows obvious differences between the pumps. In fact, the pressure behavior is so different, that using the Mahalanobis distance classifier results in perfect classification accuracy with zero false positives. ROC curves were omitted for these experiments as they are simply vertical lines at zero false positive rate ending at their maximum true positive rate.

White box modeling using the flow rate could be achieved by using the rated specifications of flow rate, or by measuring the speed at which the diaphragm of a pump alternates and estimating the volume of water each cycle displaces in order to predict the maximum flow rate. To estimate the ramps seen in Figure 4.19, one would also have to estimate the

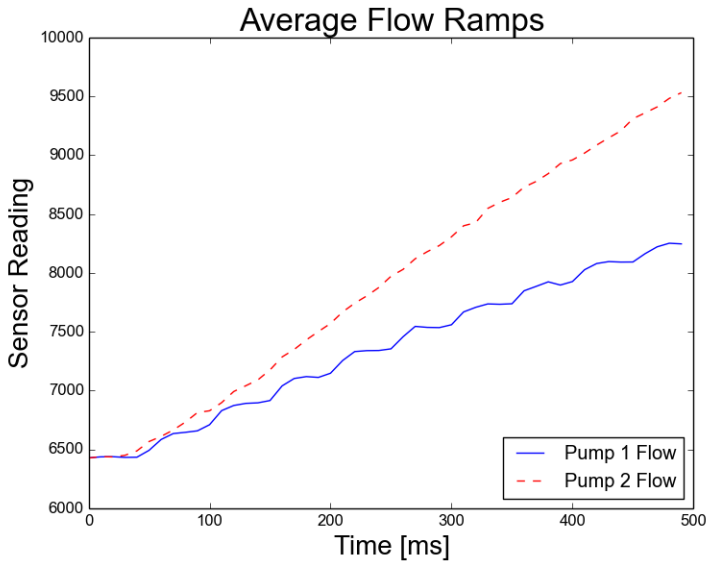


Figure 4.19: Pump flows

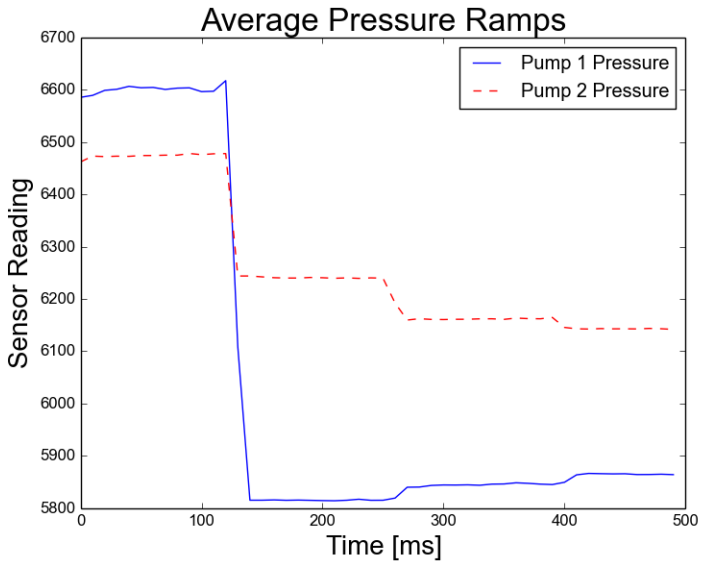


Figure 4.20: Pump pressures

effect of the paddlewheel flow meter accelerating up to the actual flow rate.

4.4 Cross-layer Response Times

The second cyber-physical fingerprinting method is evaluated using data from a live power substation and verified with controlled lab experiments. This method addresses the data acquisition half of SCADA systems by leveraging the interaction between regular polling of measurement data at the application layer with acknowledgments at the TCP layer to get an estimate of the time a device takes to process the request, and then develops a fingerprint for each device based on the distribution of these times. The timing diagram of how this measurement, which we call the cross-layer response time (CLRT), would be taken in a typical SCADA network is illustrated in Figure 4.21. It should be noted that since the CLRT measurement is based on the time between two consecutive packets from the same source to the same destination, it is independent of the round trip time between the two nodes.

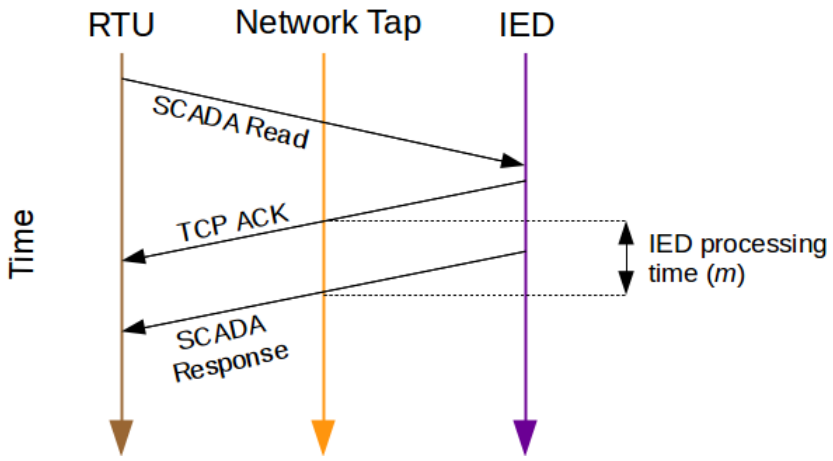


Figure 4.21: Measurement of cross-layer response time

The fingerprint signature is defined by a vector of bin counts from a histogram of CLRTs where the final bin includes all values greater than a heuristic threshold. For a formal definition, let M be a set of CLRT measurements from a specific device, B define the number of bins in the histogram (and equivalently the number of features in the signature vector),

and H signify the heuristic threshold chosen to be an estimate of the global maximum that CLRT measurements should ever take. We divide the range of possible values by thresholds t_i where $t_i = i\frac{H}{B-1}$, and define each element s_j of the signature vector by the following equation:

$$s_j = \begin{cases} |\{m : t_{j-1} \leq m < t_j, m \in M\}| & 0 < j < B \\ |\{m : m > H, m \in M\}| & j = B \end{cases} \tag{4.2}$$

4.4.1 Theory

The CLRT measurement is advantageous for fingerprinting ICS devices because it remains relatively static and its distribution is unique within device types and even software configurations. To understand why this is true for ICS devices, all of the factors which might affect this measurement must be considered.

Device Characteristics. ICS devices have simpler hardware and software architectures than general purpose computers because they are built to perform very specialized critical tasks and do little else. A typical modern-day computer now has fast multi-core processors in the range of 2-3GHz with significant caching, gigabytes of RAM, and context switching between the wide variety of processes running on the machine. In contrast, the ICS world is dominated by programmable logic controllers (PLCs) running on low powered CPUs in the tens to hundreds of MHz frequencies with little to no caching, tens to hundreds of megabytes of RAM and very few processes. With such limited computing power available, relatively small changes in programming result in observable timing differences. Depending on the desired task, different ICS device types are built with different hardware specifications (CPU frequencies, memory and bus speeds) [18] as well as different software (operating systems, protocol stack implementations, number of measurements being taken, complexity of control logic) all resulting in each one being able to process requests at different speeds. However most importantly, no matter what kind of ICS network it is in

or what physical value the device is measuring (e.g. voltage, pressure, flow rate, temperature) the device is still going to go through the same process of parsing the data request, retrieving the measurement from memory, and sending the response. Therefore, due to the limited processing power and fixed CPU load CLRTs can be leveraged to identify ICS device types, but this does not explain why the CLRTs are so constant over the network.

Network Level Characteristics. Although the use case for this technique (as in the deployment of any anomaly based IDS) would involve a training period on each target network, one of the desired properties for device fingerprinting in general is that the network architecture of the target not be a significant factor.

In a traditional corporate network mobile phones and laptops are constantly moving around and connecting to different wireless access points. The traffic they are generating is traveling over vast distances, encountering routers that are experiencing unpredictable loads, and consecutive packets are never guaranteed to take the same path over the Internet. However, devices in ICS networks are dedicated to one critical task and are fixed in a permanent location. The traffic generated from their regular polling intervals travel over relatively short geographic distance and over simple network architectures that offer little to no chance for consecutive packets to take different paths. The regular polling cycle means that routers and switches on ICS networks have consistent predictable loads which result in consistent and predictable queuing delays. Consequently for any given ICS network, a TCP ACK and SCADA response sent in quick succession will with extremely high probability take the same exact path, encounter the same delay, and therefore have a very consistent spacing in between them. Therefore, there is little opportunity for differences in network architecture to cause significant changes in the distribution of CLRTs. In Section 4.4.3 we study how much a change in networks effects the performance by learning fingerprints from one substation and testing the fingerprints over a year later on a different substation.

Due to the low computational power found in ICS devices, the CLRTs are much larger than most delays that might be caused by differences in network architecture. In the real-

world dataset used for this research, illustrated in Figure 4.24a, the CLRTs are all on the order of tens or even hundreds of milliseconds. In contrast, typical latencies obtained from ICS network switch datasheets and theoretical transmission delays on a 100Mbps link are both on the order of microseconds, resulting in a minor contribution to the overall CLRT measurement. Furthermore, ICS networks most often have over-provisioned available bandwidth to ensure reliability (e.g. the live power substation network studied for this research used an average of 11Kbps bandwidth out of the available 100Mbps, a strikingly low traffic intensity of 0.01%). These low traffic intensities ensure that the switches and routers on the network are never heavily loaded and have consistently low queuing delays.

Finally, even in the scenario where two network architectures are so different as to significantly alter the distribution of CLRTs, this would have no significant effect on the defensive utility of the proposed method and would arguably make it stronger. Any real-world application of the fingerprinting technique would involve a training period on the target network that would capture the minor effects of the network architecture. Then, if an attacker was attempting to create an offline database of signatures for all device types and software configurations without access to the specific target network, she would also have to consider all the possible network architectures that could affect them.

Due to this combination of low computational power, fixed CPU loads, and simple networks with predictable traffic, any significant change in a device's distribution of CLRTs highly suggests either an attacker spoofing the responses with a different machine, or a change in CPU workload [48] or software configuration, which could be a sign of a device being compromised with malware.

4.4.2 Experimental Setup

To test this method, experiments were run at a large scale using a real world dataset as well as on a small scale using controlled lab tests. The large scale tests were conducted in two rounds, before and after changes in the network architecture. First, network traffic (20GB)

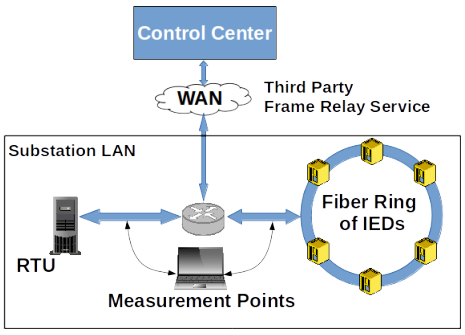


Figure 4.22: Network Architecture of First Substation

was captured from a live power substation with roughly 130 devices running the DNP3 protocol over the span of five months with the network architecture as illustrated in Figure 4.22. Then over a year later, one more month of data was captured from the same substation after the network was slightly modified by replacing the main router with a new switch, changing the IP addressing scheme accordingly, and increasing the frequency of measurement polling. Additionally, a brief overnight capture was collected from another substation with a different architecture (roughly 80 devices using DNP3, illustrated in Figure 4.23) to test if fingerprints learned on one network would translate to another. The company operating the substations provided a list of all device IP addresses on the network organized by location, device type, and device software configuration, and machine learning techniques were applied to attempt to make these labeled classifications.

Further tests were conducted in the lab to study the effects of the software configuration alone and to rule out any possible factors related to different hardware or different round-trip times (RTT) on the network.

In both scenarios, CLRT measurements were taken from DNP3 polling requests for event data and were summarized by dividing all measurements into time slices (e.g. one hour, or one day) and calculating means, variances, and 200-bin histograms for each time slice. Machine learning techniques were then evaluated using two different feature vectors: a more complex approach using the arrays of bin counts as defined in Equation 4.2 and a simple approach using arrays containing only the mean and variance for each time slice.

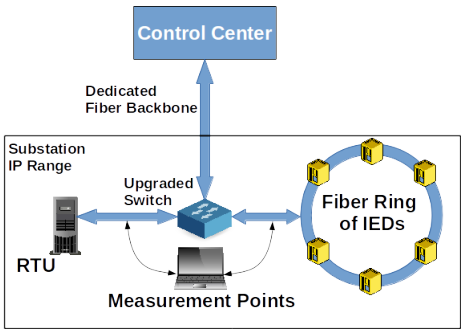


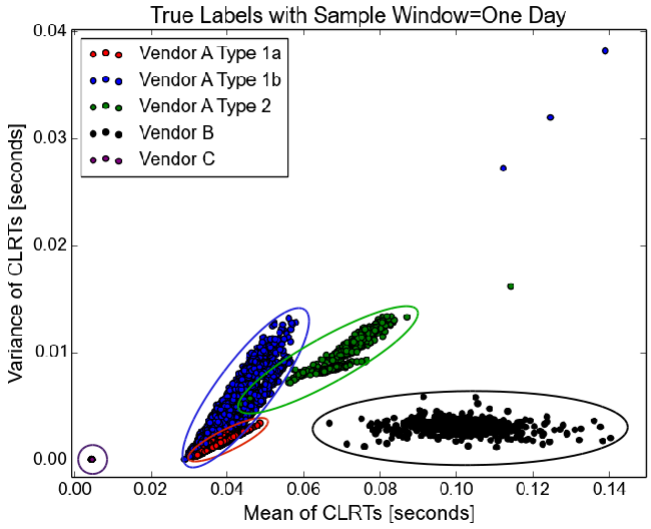
Figure 4.23: Network Architecture of Second Substation

4.4.3 Results

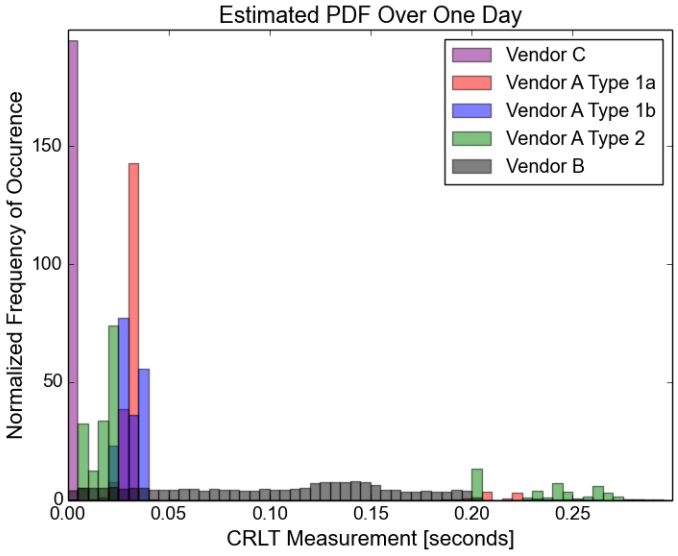
Device and Software Type Fingerprinting. To obtain a rough visualization of the separability of the device types based on their CLRT measurements, a scatter plot based on the mean and variances of CLRTs was produced and the true labels of the devices were illustrated in Figure 4.24a. Each point represents the mean and variance of the CLRT measurements for one IP address over the course of one day out of the original five month dataset. From the figure we can tell that even using simple metrics such as means and variances, results in the vendors and hardware device types being highly separable. Furthermore, it suggests that this method can also subdivide identical hardware device types into classes based on different software configurations (Vendor A Types 1a and 1b). For verification of this hypothesis, see Appendix 4.4.4. These conclusions were further supported when the probability density functions (PDFs) of CLRTs over a day were estimated for each type in Figure 4.24b.

Since Figure 4.24a illustrates that device types are clearly separable based on simple mean and variance measurements, virtually any choice of a properly tuned machine learning algorithm would result in high accuracy classification. Therefore, as the purpose and novelty of this work is not the use of machine learning for fingerprinting, a sampling of the most popular algorithms in the field were chosen as examples.

To measure the performance of our fingerprinting techniques throughout this work, we calculate the standard classification metrics of accuracy, precision, and recall as defined in



(a) CLRT samples for all devices with a time slice of one day



(b) Estimated PDFs of CLRTs for five sample devices over one day

Figure 4.24: Separability of device types based on CLRT

Equations 4.3, 4.4, and 4.5 for each class separately, where TP, TN, FP, and FN stand for true positive, true negative, false positive, and false negative, respectively. To summarize these results, the average value across classes was plotted alongside the minimum value among classes.

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.3}$$

$$PREC = \frac{TP}{TP + FP} \tag{4.4}$$

$$REC = \frac{TP}{TP + FN} \tag{4.5}$$

The first machine learning algorithm used in these experiments to classify the labeled data was a feed forward artificial neural network (FF-ANN) with one hidden layer trained using the back propagation algorithm. This algorithm was chosen due to its popularity and previous use in related work [18]. The bin counts of the histograms, as defined in Equation 4.2, were used as the feature vector for each sample and the time slice they were taken over was varied. The samples were randomly divided using 75% as training data and 25% as testing data. The average and minimum accuracy, precision, and recall for these experiments are shown in Figure 4.25, and suggests that even with time slices as small as 5 minutes an average accuracy of 93% can be achieved. Some devices at this substation were being polled only once every 2 minutes, so the 5 minute detection time is roughly equivalent to a decision after only two samples. Furthermore, when false data is injected into a control system catastrophic damage usually cannot immediately occur due to built-in safety features in the system. The most successful attacks would sabotage equipment or product over an extended period of time, for example by tricking a control system into heating a reactor past its limits and causing it to explode.

To demonstrate that the exact choice of machine learning algorithm is largely irrelevant, we also attempted supervised learning using one of the simplest algorithms in the literature,

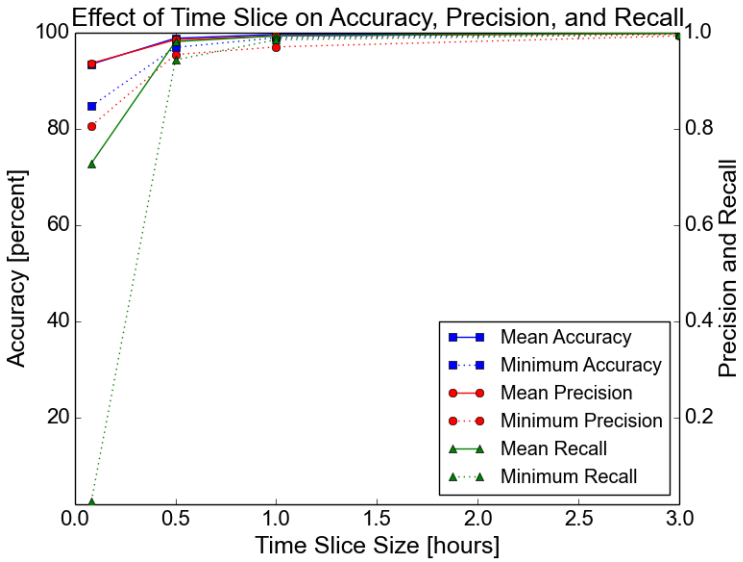
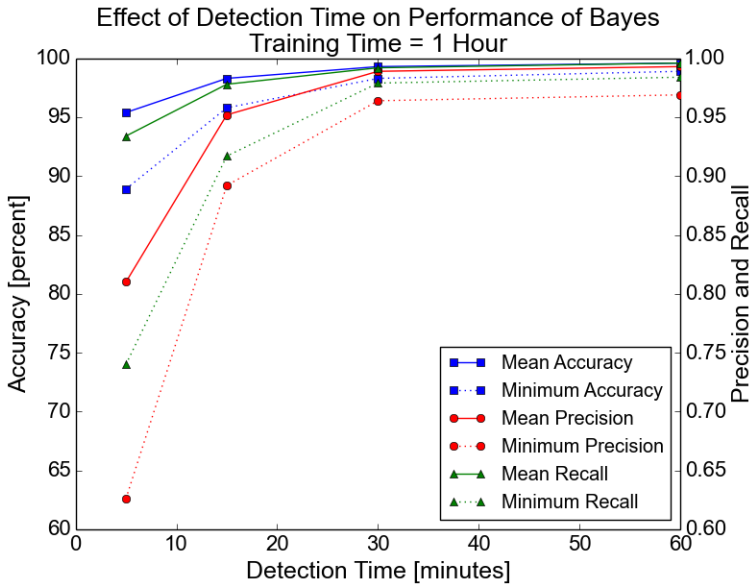


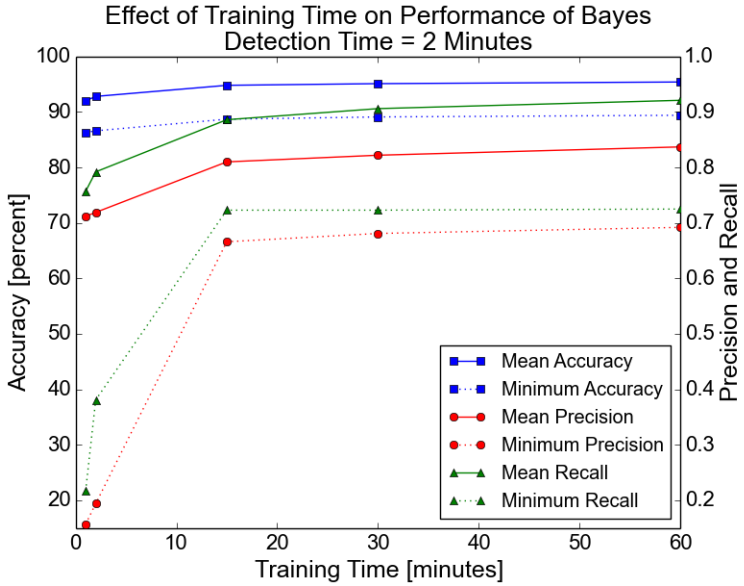
Figure 4.25: Fingerprint Classification Performance Using FF-ANN

a multinomial naïve Bayes classifier. The signature vectors remained the same and similar experiments were conducted to determine the required training period and detection time. Furthermore, these tests were conducted to simulate a real world deployment instead of randomly choosing training and test data, the training data was taken from the beginning of the capture and the test data was taken from the following 1000 detection time windows. After studying Figures 4.26a and 4.26b, it is clear that the simple Bayes classifier performs even better than the more complex ANN and can achieve high accuracy classification with detection times as small as a few minutes.

The above results are extremely promising for supervised learning when a list of IP addresses and corresponding device types are available, but this is not the case for administrators trying to understand what devices are on a poorly documented legacy network. To address this scenario, unsupervised learning techniques were also applied and tested if they could accurately cluster the devices into their true classes. Referring back to Figure 4.24a, it is clear that the samples closely follow a multivariate Gaussian distribution, so it was decided to illustrate unsupervised learning with Gaussian mixture models (GMM) using a full covariance matrix and a signature vector consisting of means and variances with a time



(a) Accuracy, precision, and recall of supervised Bayes classifier as a function of detection time



(b) Accuracy, precision, and recall as a function of training time

Figure 4.26: Fingerprint classification performance

slice of one day. Figure 4.27 shows the estimated clusters learned from the GMM algorithm, which upon comparison with the true clusters in Figure 4.24a, looks very similar. When the dataset was tested against the learned clusters, the model achieved an accuracy of 92.86%, a precision of 0.891, and a recall of 0.956. With performance as nearly as high as the supervised learning methods, this unsupervised technique would allow administrators to develop an accurate database of fingerprints with very little knowledge of the network itself.

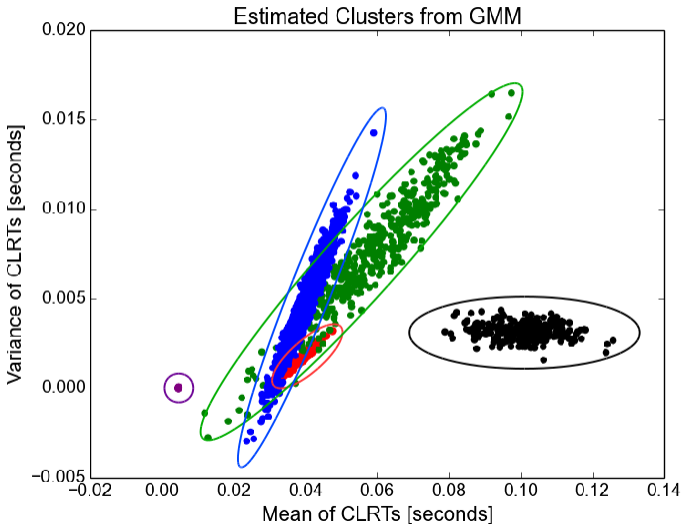


Figure 4.27: Randomly generated samples from the unsupervised learned clusters

Effect of Network Architecture. While the previous experiments, simulating a real-world deployment with a training period on the target network, performed very well, we also wanted to study how much the network architecture affects the performance of the fingerprinting techniques. For the first experiment to study these effects, the original substation was revisited over a year later after the network architecture had been upgraded and polling frequency had been increased. When the distribution of the new architecture in Figure 4.28a is compared with the original in Figure 4.24b, there are only minor differences. When the fingerprints learned from the original capture were tested on the new data, very high accuracies in Figure 4.29a were obtained suggesting that the method is stable over

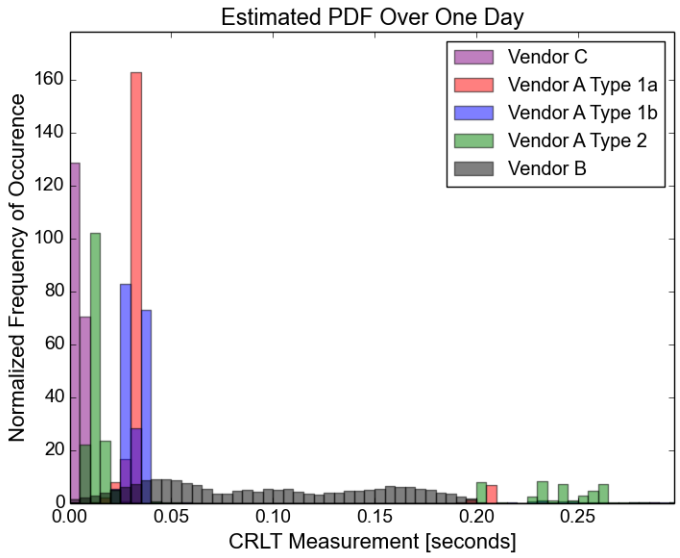
long periods of time and over minor changes in the same network.

Even though the primary defensive use-case for this technique would always involve a training period on the target network, we also consider the rare case where an administrator is able to learn fingerprints on one network because of known labels, but does not have the labels for a different network. To study this scenario, we learned fingerprints from our original capture and tested them on a different substation over a year later. When the different substation's distribution in Figure 4.28b is compared with the original there are some small, but noticeable changes that could be result of the different architecture affecting the timings or from the different electrical circuit affecting the load of the devices. When the fingerprints learned from the original capture were tested on this different network, the average accuracy seemed to level off around 90% suggesting that while the accuracy may be diminished across different networks, there is still some utility in the technique.

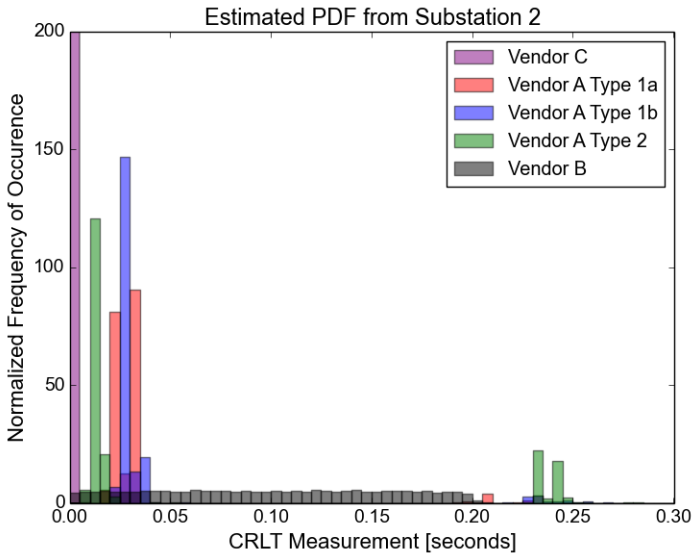
Finally, to show that the technique performs well on different networks when trained individually, we trained a Bayes classifier on one hour of data from the second substation and tested it on the remaining seventeen hours of data with the results in Figure 4.30.

4.4.4 Software Configuration Fingerprinting

To verify the suggestions from the large scale experiments that the software configuration was observable through CLRT measurements, lab experiments were performed on the same exact IED with different settings enabled and disabled. Approximately 700 CLRT measurements were taken for each of three cases: all extra settings enabled, only overcurrent protection enabled, and all extra settings disabled. When comparing the distributions for all extra settings enabled versus disabled in Figure 4.31, there are several noticeable differences. In fact, when the same FF-ANN from the previous experiments was trained on these two cases, perfect classification accuracy was achieved. Figure 4.32 shows only minor differences between the 'free' case and the overcurrent case, and consequently, the FF-ANN only achieves roughly 66% classification accuracy.

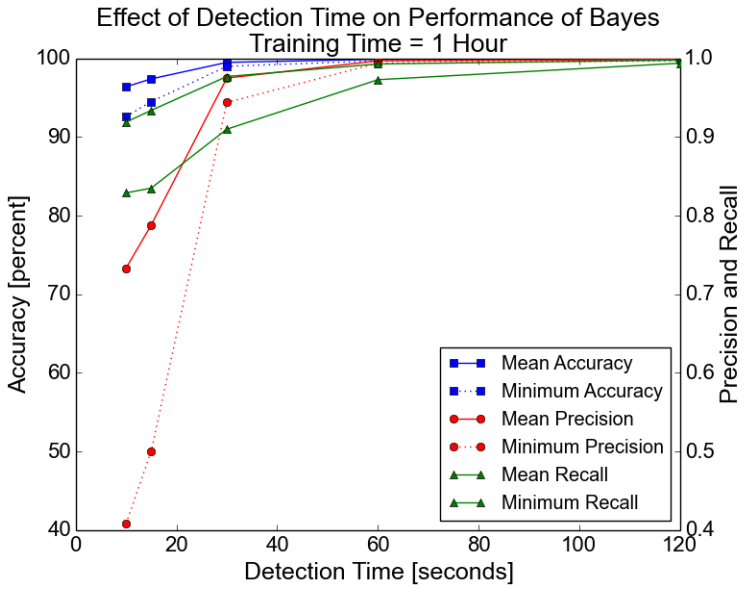


(a) CLRT Distribution After network Changes

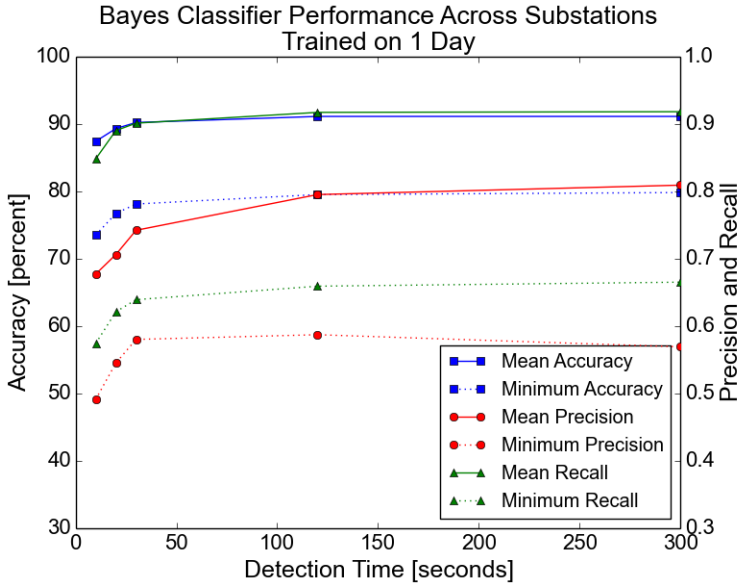


(b) CLRT Distribution of Second Substation

Figure 4.28: Minor effects of network architecture on CLRT distributions



(a) Classification Performance After Network Changes



(b) Classification Performance Across Substations

Figure 4.29: Classification Performance Across Networks

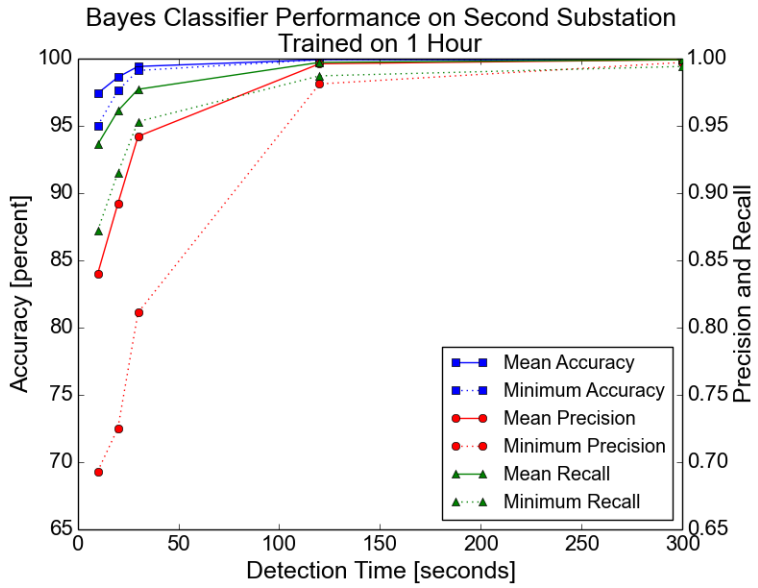


Figure 4.30: Classification Performance on Second Substation

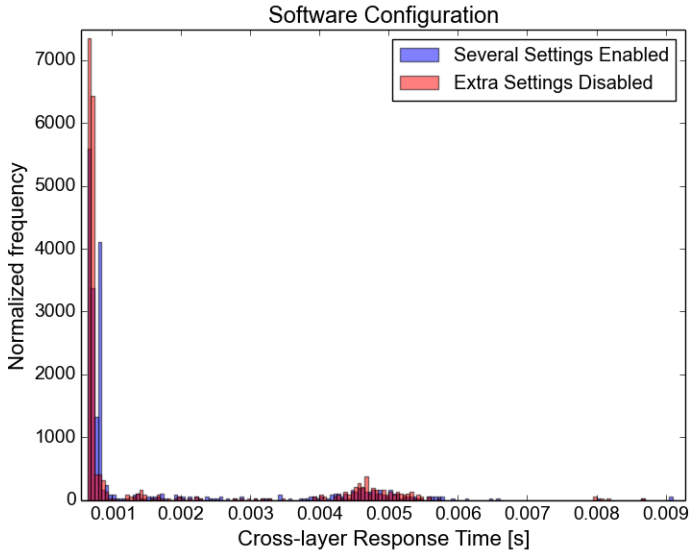


Figure 4.31: Effect of multiple settings enabled on CLRT distribution

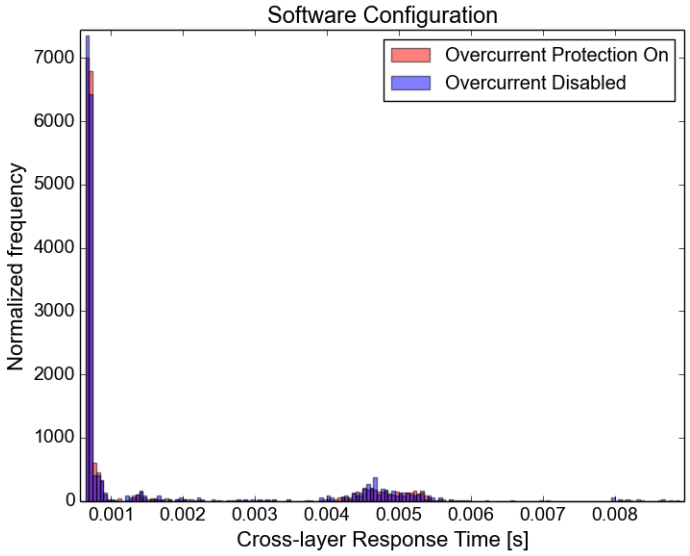


Figure 4.32: Effect of one extra setting enabled on CLRT distribution

4.5 Discussion

4.5.1 Performance

In order for a fingerprinting method to be useful for any situation, whether it is for intrusion detection, surveillance, or network management, the techniques should be accurate and scalable.

Accuracy Effective intrusion detection techniques require high true positive rates (TPRs) with acceptable false positive rates (FPRs). The definition of what an acceptable rate of false positives is depends on the base rate of classification events. In the IT world, a network can see on the order of a million "events" per day, so the false positive rate must be on the order of 10^{-5} in order to achieve a manageable number of false positives per day. In the ICS world, the base rate of physical events is much lower. A given local ICS network may have a few hundred physical devices, with each being operated only a handful of times per day resulting in a base rate on the order of a thousand events per day. With this base rate, acceptable FPRs only need to be on the order of $10-2$ to be manageable. While the performance of the previous physical fingerprinting research was borderline acceptable (92%), the new methods dramatically improve the accuracy. Perfect classification performance was achieved for pumps, valves, and relays, and three-class AC motor classification performed slightly worse with a TPR of 88% and FPR of 5%. Again, note however, that these results are expected to improve in a real-world deployment where different motors are driving different loads.

For the CLRT method, the relevant base rate to consider is the number of read requests on the network. For the given test data, around 100 nodes were on each network, being polled roughly once a minute, resulting in a base rate on the order of a few hundred thousand events per day and a target FPR on the order of 10^{-4}. The CLRT method achieved classification accuracies as high as 99% in some cases, providing acceptable performance for input into an IDS, but not ideal. False positives could be further reduced using alert

correlation or by incorporating a confidence threshold before alerting.

Scalability The FF-ANN algorithm used in training the two fingerprinting techniques only had one hidden layer and 200 input features, resulting in reasonable scalability for computational complexity. The other methods including the Bayes classifier and Mahalanobis distance classifier are standard techniques that are computationally efficient as well. Furthermore, our results suggest that the accuracy for the methods scales as well. The CLRT method was already tested above on a full scale power substation network and was able to achieve high accuracies.

4.5.2 Robustness Against Forgery

When using device fingerprinting to augment traditional IDS methods, it is also desired that the fingerprints be non-trivial to forge (i.e., resistant to mimicry attacks). Fortunately there are several reasons as to why the proposed methods are not so easily broken. First, there is always going to be inherent randomness in the attacker's machine that makes it non-trivial to perfectly reproduce anything based on precision timing, and difficult to accurately reproduce the CLRT fingerprint. The previous physical fingerprinting work had accuracy limitations from using network timing and clock synchronization, but these same aspect made it difficult for physical fingerprints to be forged.

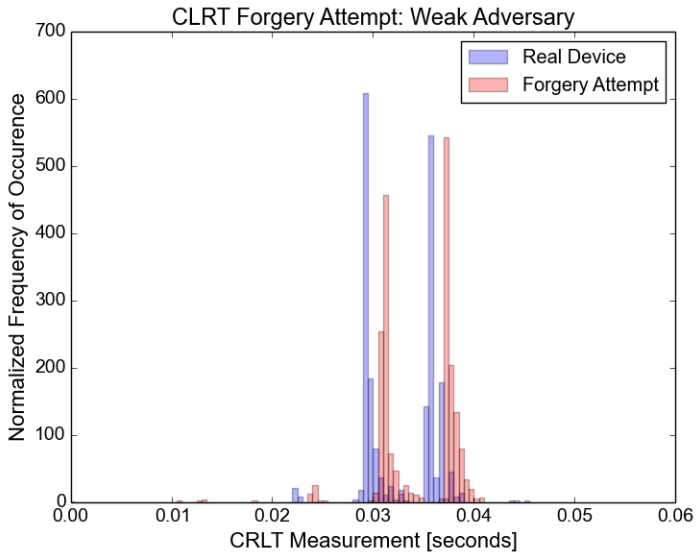
To evaluate the proposed methods against forgery, we consider two different classes of adversary. First, we consider the case where an adversary is unable to gain physical access to the target network but instead is able to compromise one of the low powered devices on an air-gapped network, as in the case of Stuxnet [42]. Her goal is to watch the network long enough to generate black box fingerprints and spoof the responses of another device while matching their fingerprint. To model this adversary, we use a BeagleBone Black with 512MB of RAM and its ARM processor clocked down to 300MHz to simulate the resources available on a high-end PLC. Second, we consider a stronger adversary that has gained physical access to the network and is able to use her own, more powerful, machine to spoof

the responses. This stronger adversary was modeled by a standard desktop with a 3.4 GHz quad-core i7 processor and 16GB of RAM. In both scenarios, the adversary is assumed to have gathered accurate samples and therefore has perfect knowledge of the signature she must try to mimic. However, in reality there are several difficulties that would make this perfect knowledge unlikely. First, since the ICS environment contains an abundance of legacy devices, it is not certain that the compromised device would even have a network card that supports promiscuous mode for network sniffing. Additionally, any sniffing code installed on a low powered, compromised device would most likely be computationally expensive enough to skew timing measurements on the system. Furthermore, since it was found in Figure 4.29 that network architecture does have some effect on the fingerprint, this suggests that the adversary would have to sniff the network in the same location as the fingerprinter to get a completely accurate distribution, or be able to determine the effects of the network by other means.

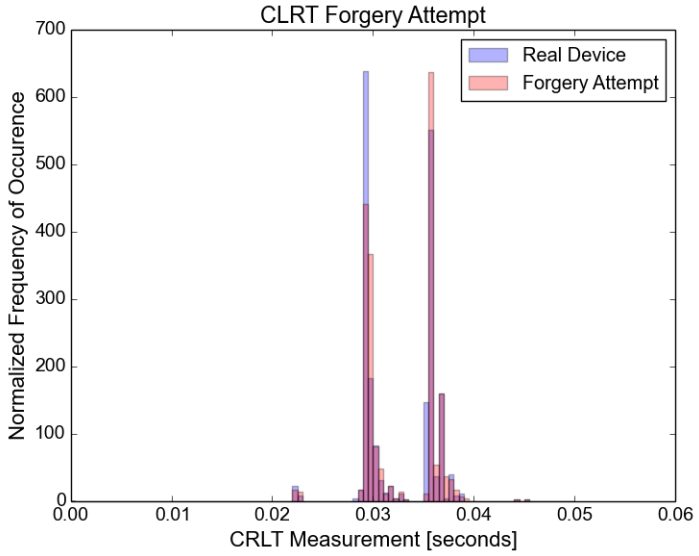
Cross-Layer Response Time Forgery

To test the cross-layer fingerprinting method, an open source implementation of DNP3 (OpenDNP3 version 2.0.1) was modified to have microsecond precision sleep statements using the known CLRT distribution of one of the Vendor A Type 1b devices. The forgery attempt by the weaker adversary in Figure 4.33a shows very clear differences in the distributions due to the limited resources slowing the distribution down and adding its own randomness. The stronger adversary's forgery attempt can be seen in Figure 4.33b. Compared with the original, the two distributions are very similar but the forged one is slightly slower due to the adversary's own processing time.

When the Bayes classifier was applied to distinguish between the real device's distribution and the attacker's forged distribution, the results in Figure 4.35 suggest high accuracy detection of the forgery can be achieved.



(a) Forgery Attempt for CLRT Fingerprinting Under Weak Adversary



(b) Forgery Attempt for CLRT Fingerprinting Under Strong Adversary

Figure 4.33: Forgery attempts against the CLRT technique

EPF Forgery

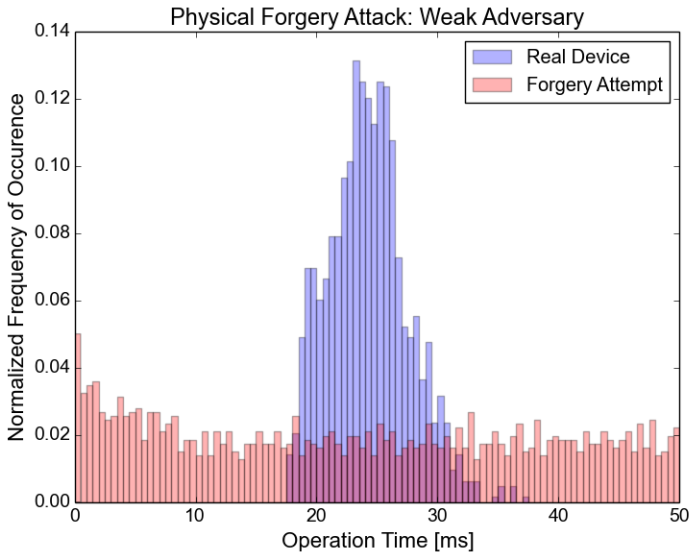
To study the forgery of the EPF physical fingerprinting technique, a DNP3 master was configured to send operate commands every second, and the adversary machine's modified OpenDNP3 code was programmed to send responses with timestamps calculated from the machine's current time, added with the known distribution of operation times. The resulting forgery attempt by the weaker adversary can be seen in Figure 4.34a. The distributions appear completely different due to the BeagleBone's clock quickly drifting from the SCADA master's, thus making the forgery attempt easily detected. The forgery attempt by the stronger adversary, illustrated in Figure 4.34b, is similar to the original, but still has noticeable differences most likely due to the high-end PC timestamping the operations faster than the original device.

The results from the Bayes classifier in this scenario in Figure 4.35 also suggest high accuracy detection of forgery is possible.

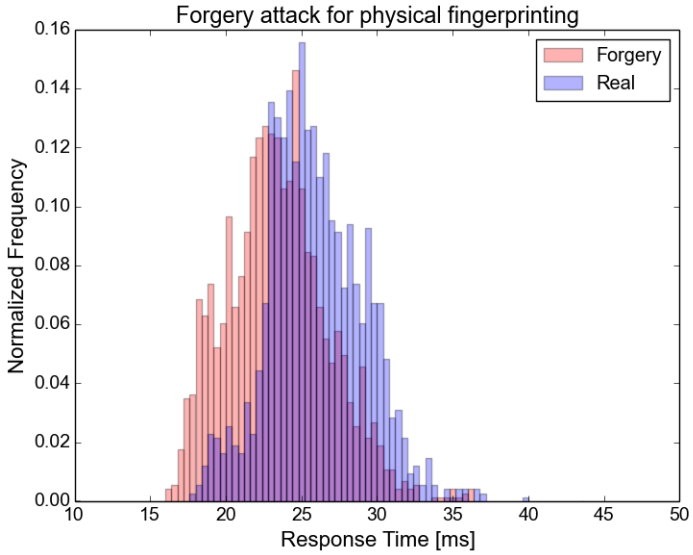
Even though both the CLRT and EPF fingerprinting techniques exhibit resistance to these naïve forgery attacks, it is still theoretically possible that an attacker could more intelligently shape her response times to more closely match the true fingerprint and implement a method of keeping better clock synchronization with the target. However, this would require a *significantly* more knowledgeable and skilled adversary to successfully accomplish. She would have to know beforehand the relative speed of her machine to the target's machine, have knowledge of any effects the network architecture might have on the signature, and determine how fast the target's clock drifts, all suggesting that these methods are robust enough to be used as part of a defense-in-breadth IDS strategy.

BPF Forgery

Unfortunately, with the movement of the operation time measurement from the network to the local PLC, there is a tradeoff with robustness. As a reminder, in the extended physical fingerprinting techniques, the PLC itself is measuring the operation time and reporting it

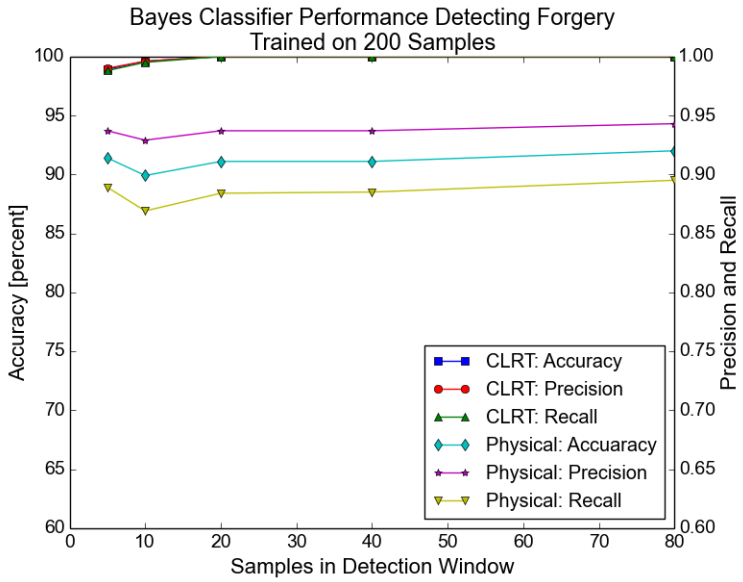


(a) Forgery Attempt for Physical Fingerprinting Under Weak Adversary

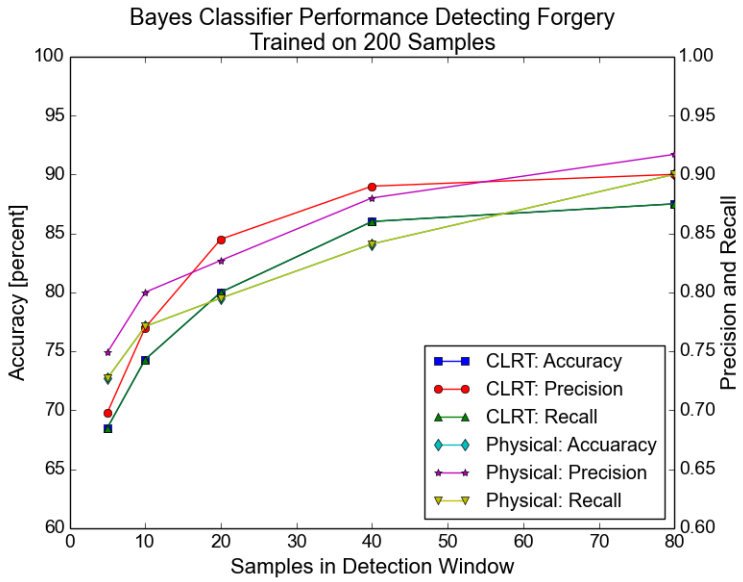


(b) Forgery Attempt for Physical Fingerprinting Under Strong Adversary

Figure 4.34: Forgery attempts against the physical fingerprinting technique



(a) Forgery Detection for Weak Adversary



(b) Forgery Detection for Strong Adversary

Figure 4.35: Forgery Detection

back over the network using standard ICS network protocols, which are notorious for not providing any encryption or integrity mechanisms. This means that any adversary who can sniff the network can then have full knowledge of the distribution and replay a perfect forgery over the network. Even though there are several practical obstacles to an adversary gaining the necessary access to a device capable of sniffing the network, this provides no security guarantees and only increases the difficulty of successfully injecting the false data.

4.5.3 Limitations

While both proposed fingerprinting methods perform well under certain conditions, there are some limitations. The cross-layer fingerprinting method first requires a SCADA protocol using "Read" and "Response" messages, which all of the most popular SCADA protocols implement. Furthermore, the SCADA protocol must sit on top of a TCP implementation that uses at least a minimum amount of "quick ACKs" (immediately ACKing a packet instead of delaying in the hopes of piggybacking). For example, modern Linux systems use quick ACKs to accelerate TCP slow start at the beginning of connections and after retransmissions, but every vendor in the observed power substation dataset used quick ACKs for every packet, presumably to reduce latency. Therefore, the amount of quick ACKs used by a device would determine how quickly a fingerprint could be generated.

The new BPF physical fingerprinting techniques vastly improve on the accuracy and scalability of the previous EPF methods, however at a cost in robustness.

The highest classification accuracies achieved in the CLRT work (around 99%) are impressive but would result in an impractical number of false alarms (1%) if each misclassification was treated directly as an intrusion. Similarly the new BPF physical fingerprinting work, while 100% accurate for the tested pumps and relays, achieved only an 88% true positive rate with a false positive rate of 5% for motors. Therefore, any practical application of these fingerprinting techniques to detect intrusions would leverage the significant body of work [49] [50] on IDS alert correlation to manage the number of alarms.

4.6 Conclusions

In this chapter we presented two novel methods for passively fingerprinting devices on ICS networks. After evaluating the methods using real world datasets and controlled lab experiments, fingerprint classification accuracies ranged from 88% to perfect 100%. Both techniques provide no security guarantees, but significantly increase the difficulty for an adversary to successfully attack the system, providing important input into a real-world ICS intrusion detection system.

Combining the RTT observations from Chapter 3 and the CLRT methods here suggests that embedded devices like those in ICS networks are uniquely well suited for fingerprinting techniques based on their processing and execution times. The next chapter extends this idea to fingerprint the programs running on PLCs from their program execution times.

CHAPTER 5
PLC PROGRAM FINGERPRINTING

5.1 Introduction

Programmable logic controllers (PLCs) interface the cyber world with the physical world, continuously executing their control program scan cycles of reading physical inputs, executing control logic, and updating physical outputs such as motors, valves, and pumps to control the power grid, manufacturing, and chemical processing facilities. Unfortunately, as critical as these devices are, they are still being sold insecure by design. Not only are vulnerabilities being found at increasing rates in related software [51], but vendors still struggle to implement the most basic security functions like passwords and message authentication. With strict downtime requirements discouraging frequent patching, limited resources prohibiting strong cryptography, and vendors who are slow to respond to obvious security issues, these vulnerabilities will be solved any time soon. To secure these networks before an attack on an ICS results in the first cyber related loss of life, it is necessary to build strong scalable techniques for detecting intrusions on vulnerable ICS networks.

To help address this problem, techniques were developed to detect changes to PLC programming at the application layer. The application layer represents one of the easiest vectors for attackers to stealthily cause physical damage to the victim facility, and in fact is the only method used so far in real-world attacks, namely Stuxnet [42]. It was again used in the theoretical ransomware attacks demonstrated on PLCs at the RSA Conference this year [52] [53]. The unique aspects of PLCs including the low-powered hardware, real-time operating systems, and deterministic cyclic execution of the control programs combine to allow accurate fingerprinting of the current program running on the device. By monitoring the execution times of the program running on the PLC, changes to the program

can be detected with high accuracy and low false positives. These execution times can be monitored from the underlying diagnostic connection, or by adding just a few lines of code to the PLC program to gather more accurate measurements. Both of these approaches *require no modifications to the underlying firmware, no action by the vendors of the devices, and can be immediately implemented by ICS operators*.

The significant contributions of this research include:

- The first comprehensive measurement of the effects of program changes on PLC scan cycle times
- Improved methods for more accurate measurement of scan cycle times when diagnostic cycle time resolution is coarse
- Novel techniques for detecting PLC program modifications from both the network *and* local access using cycle time signatures
- White box modeling approaches for estimating scan cycle time signatures from source code, for rare execution branches
- Unique application of proof-of-work functions to provide intrusion detection mechanisms for low powered real-time devices

The remainder of this chapter is organized as follows. Relevant background information on the design and operation of PLCS is given in Section 5.2. The threat model addressed and assumptions made in this research are explained in Section 5.3 and details about the experimental setup are given in Section 5.4. Section 5.5 addresses a low skilled attacker model and measures the effects different instructions have on scan cycle times and how accurately scan cycle times can be used to detect added instructions. These basic observations and techniques are then extended with white box analysis for rare branching conditions. Change detection algorithms are applied in Section 5.6 to detect a medium skill adversary adding the minimum amount of instructions necessary for a logic bomb trigger.

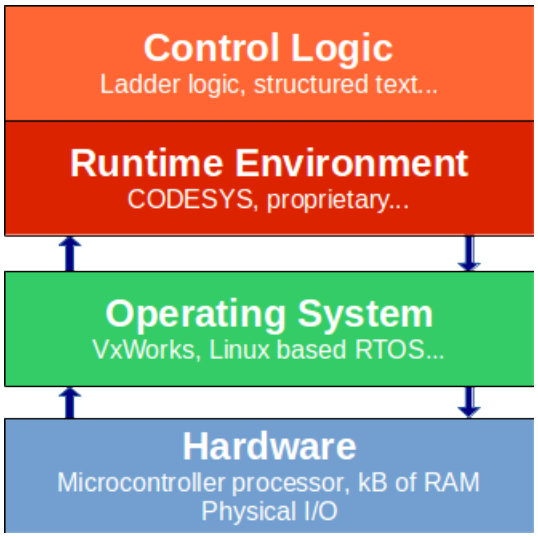


Figure 5.1: Architecture of a PLC

Finally, Section 5.7 addresses a more intelligent attacker model by applying proof-of-work functions as a means of more robustly detecting added instructions to the program. Overall performance in terms of accuracy and security against certain attacks are discussed in Section 5.8 and final conclusions of this chapter are drawn in Section 5.9.

5.2 Background

To understand the techniques proposed in this chapter, first a general overview of PLC operation and architecture is helpful. PLCs are essentially industrialized microcontrollers, providing the bridge between the cyber and physical worlds by controlling devices such as valves, pumps, and motors in response to operator input or their preprogrammed control logic. They are used in a wide range of applications including the control of power generation, water treatment and distribution, various stages in the oil and gas vertical, and even for smaller automation tasks such as elevators, escalators, and roller coasters. PLCs in general follow the architecture illustrated in Figure 5.1, with resource limited processors at the bottom layer. As an example, the highest end PLC model being sold today from Schneider Modicon boasts a 266 MHz processor with 11 MB of RAM with costs ranging upwards of twenty thousand dollars [54].

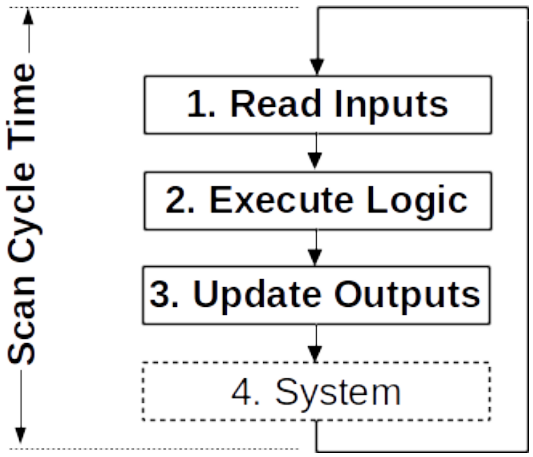


Figure 5.2: PLC Scan Cycle Times

On top of the low powered hardware, the PLC typically runs some form of Linux based real-time operating system, such as VxWorks, to provide deterministic guarantees of program execution times. At the application layer of the PLC sits the runtime environment which the user interfaces with to load the desired control logic program. The control logic program is executed inside an endless loop, illustrated in Figure 5.2, with the PLC continuously reading physical inputs, executing the control program, updating physical outputs, and performing system functions such as network communication. The runtime environment monitors the execution of the control program and provides diagnostic information such as CPU and memory usage as well as the PLC scan cycle time, typically defined as the time it takes for one whole cycle of Figure 5.2. However, some vendors measure the scan cycle time only by the control logic execution times.

5.3 Threat Model and Assumptions

The threat model this research addresses is that of an adversary of varying skill levels whose goal is to maliciously reprogram a PLC at the user application layer. This threat model was chosen, and not one where an adversary has compromised the underlying operating system, because it mirrors the only known attack on PLCs to date and is in fact the most feasible scenario for a successful attack.

Historical. To date, the only publicly known real-world cyber attacks that achieved a physical impact on ICS networks followed this threat model. Stuxnet, as complex of an attack as it was and skilled as the adversaries behind it were, did not exploit any vulnerabilities on the Siemens PLCs themselves, but rather used a compromised engineering workstation (a PC on the ICS network) to simply reprogram the PLCs with malicious control logic. Additionally, the Ukrainian power outages were not caused by zero-day exploits on the circuit breaker equipment, but again simply used a compromised workstation to send legitimate commands [55]. Finally, the recent demonstration of a ransomware attack at the 2017 RSA Conference only attacked the application layer of the PLCs, reprogramming them with malicious control logic [52] [53].

Feasibility. Attackers, even with nation-state level skill, tend to take the path of least resistance in achieving their goals. Unless the target PLCs are directly facing the Internet, the path an adversary must take to pivot down to the PLC level most likely would use an engineering workstation as a stepping stone. If the attacker has already compromised such a system, it is much easier for him to use the legitimate machine and software to achieve his goals stealthily than to launch an exploit. By choosing the attacker model used here, we force the attacker to add yet another step in his attack chain. This not only increases the time and resources he must spend, deterring cybercriminal actors, but also increases the complexity of the attack and thus the chances the attacker will make a detectable mistake.

The attacker is assumed to be able to observe all traffic and scan cycle times being sent over the network and to have full knowledge of the proposed detection methods. However, given the stated assumption that the attacker is only modifying the user application layer, the diagnostic information coming from the underlying system can only be indirectly modified by shaping the control program. Low skilled attackers are addressed first, in a scenario where the PLCs are reprogrammed with completely different functionality. Then medium skilled attackers are addressed simulating scenarios where the attacker attempts to insert the smallest logic bomb possible in the program, but is not knowledgeable enough to

spoof the reported scan cycle times. Finally, high skilled attackers are addressed with the inclusion of proof-of-work functions to make it infeasible for attackers to mimic the scan cycle signature without removing instructions from the original program.

5.4 Experimental Setup

The devices used in this research come from three of the most popular PLC vendors: Siemens, Allen Bradley, and Schneider Modicon, who combined hold the majority market share in PLCs. Relevant information about each model is summarized in Table 5.1, including the memory available for user programs and the resolution that diagnostic connections give for scan cycle times. The application memory limits an adversary's ability to precompute and store proof-of-work results and the cycle time resolution limits how small of a change the adversary can make undetected. Measurement techniques of the scan cycle times are demonstrated first using this system level diagnostic information, and then later more accurate and robust methods for measuring scan cycle times are proposed in Section 5.5.3.

Table 5.1: Overview of Devices and System Information

PLC Model	Application Memory	Cycle Resolution
MicroLogix 1100	8 KB	100 μs
Siemens S7-1200	75 KB	1 ms
Schneider M221	256 KB	1 μs
Schneider M241	8 MB	1 μs

It is difficult to obtain a significant body of large scale example PLC programs due to the non standard program formats and the fact that the system integrators writing the programs often consider them proprietary intellectual property. However, a small body of real world programs were gathered from the "Mr. PLC" forum website [56] where users have posted example code from real projects to help others learn. The top four most viewed PLC programs for Allen Bradley PLCs (excluding outliers that were extremely simple or extremely complex) were manually ported to all the PLCs in Table 5.1. The

general description of the programs are summarized in Table 5.2, with their complexity, according to the MicroLogix 1100.

To measure the effects of small program changes on scan cycle times, programs were written for all PLCs that tested increasing numbers of the most common instructions such as timer blocks, counter blocks, examine if closed (XIC), and move operations. Additionally, the example programs from Table 5.2 were modified by adding the smallest amount of instructions possible for an attacker to implement a logic bomb in different ways. These included the addition of a timer, a counter, a single compare operation, and a single XIC instruction. For all experiments, unless otherwise noted, a total of 10,000 scan cycle samples were taken at a rate of 10 samples per second totaling over a 16 minute duration.

Table 5.2: Complexity of Programs Tested

Program	Description	Instructions	Data Words
P1	Motor Starter	553	1068
P2	Sequencer Example	365	160
P3	Bottling Plant	419	433
P4	Conveyor Belt	615	425

5.5 Change Measurement

At the most basic level, the foundation of this research is based on the idea that any change to the control logic program will produce some change in the program execution time. This section explores the intuitive theory behind why this is especially true for PLCs and provides detailed measurements of how the execution times are effected by different instructions.

5.5.1 Theory

There are important differences, both in their design and usage, between PLCs and personal computers (PCs) that make the use of execution times a powerful basis for intrusion detection for one and not the other.

Modern PCs are built on Gigahertz processors where single instructions take mere nanoseconds to execute, which is infeasible to accurately measure. PLCs are built on much slower microcontroller processors typically on the order of tens of MHz. Assuming a fully pipelined processor running at 10MHz a single instruction on the CPU takes a measurable length of time at $0.1\mu s$. The operating systems typical today on PCs treat all user processes roughly the same when scheduling them, with run times varying widely depending on the number of times a program was interrupted by other processes. PLCs on the other hand are designed with real-time operating systems that guarantee deterministic scheduling of high priority processes. The number and computational burden of the programs running on PCs also varies widely further contributing to unpredictability in program execution times. Again, the case is different for PLCs where a constant number of processes are running and there is little to no human interaction to introduce sporadic CPU loads.

5.5.2 Baseline Measurements

As a first pass, we address the least skilled attacker model where the attacker reprograms the PLC with completely different functionality without trying to match any signatures. For a simple illustration of the differences in scan cycle times this would cause, consider the PLCs and example programs described in Section 5.4. All programs were similar in complexity, with hundreds of instructions, and all PLCs were in the same price range being lower end models less than $1000 each. For this first set of experiments, program execution times were gathered for all devices using their own standard diagnostic connection, with different vendors offering different resolution, as listed in Table 5.1.

For the Modicon M221, the system resolution of the scan cycle time was given in microseconds. In Figure 5.3, all programs exhibited similar ranges of standard deviations in scan cycle times, but had consistent differences in mean execution times. Even the two closest programs, the conveyor belt and sequencer example, consistently had more than a 2 μs difference.

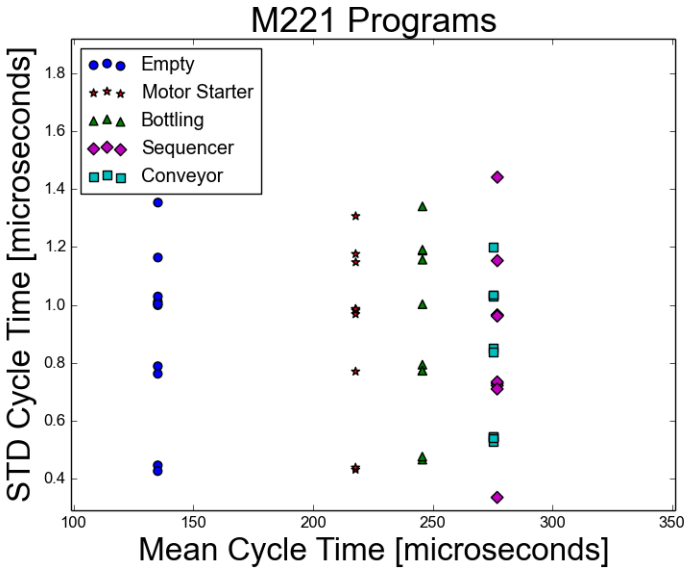


Figure 5.3: Modicon M221 Program Execution Times, resolution in μs

The Modicon M241, a more expensive model, also provides microsecond level precision for scan cycle measurements and exhibits even more clear differences in Figure 5.4. For the Allen Bradley MicroLogix 1100, the diagnostic information only provides scan cycle time resolutions of $100\mu s$, resulting in much more overlap in Figure 5.5.

Finally, for the Siemens S7-1200, the system level diagnostic information only provides scan cycle measurements at a $1ms$ resolution. Given that all programs tested were relatively small, no program execution time exceeded a mean time of $1ms$ so such diagnostic information is useless for detailed classification. However, the underlying theory that any change to the control program will affect the program execution time still holds, and this observation simply motivates the development of a more accurate method for measuring scan cycle times.

5.5.3 Improved Scan Cycle Measurement Methods

Given the low resolution of the MicroLogix 1100 and S7-1200 diagnostic level measurements of scan cycle times, example programs exhibited little to no clear classifiable differences. However, if modifications are made to the PLC's control program, more accurate

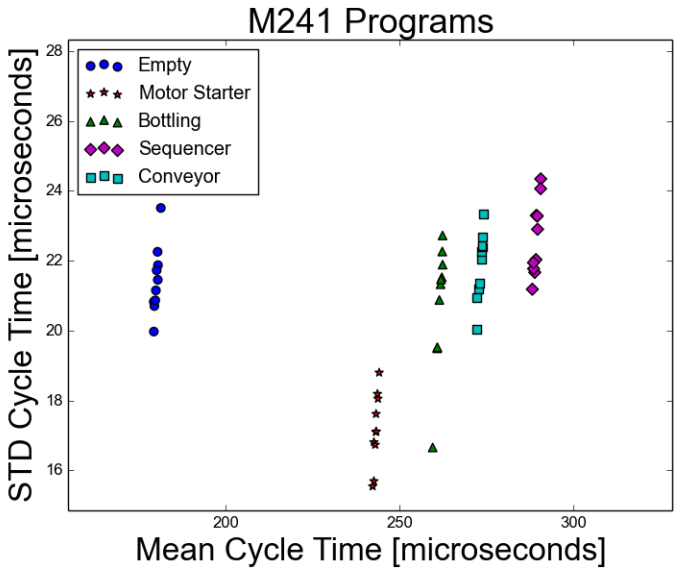


Figure 5.4: Modicon M241 Program Execution Times, resolution in μs

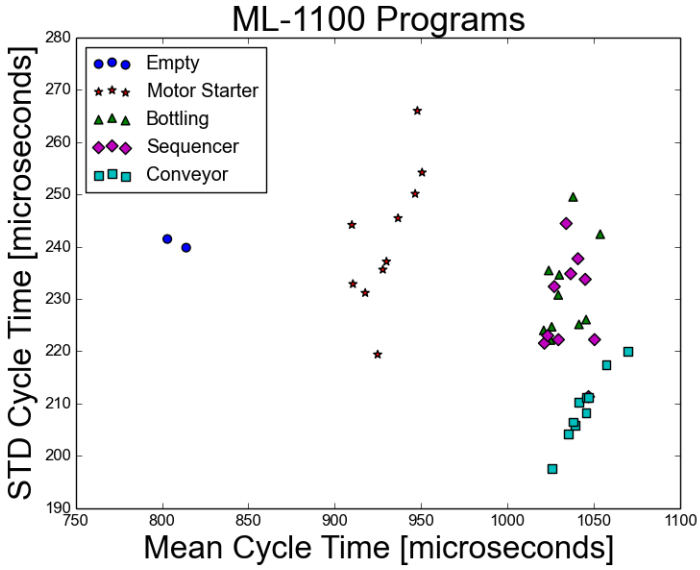


Figure 5.5: MicroLogix 1100 Program Execution Times, resolution in $100\mu s$

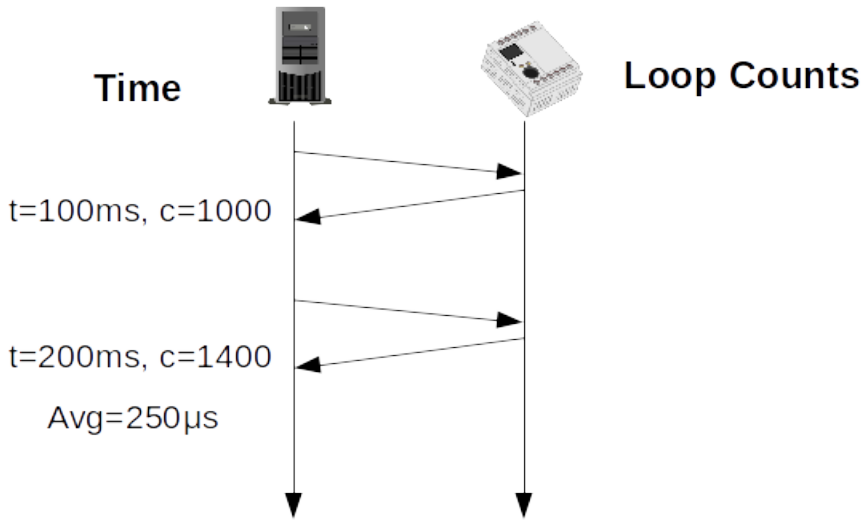


Figure 5.6: Method for more accurate measure of scan cycle times

measurements can be taken.

Given the understanding that PLC control programs are essentially executed in an endless loop as fast as possible, by measuring the number of loops executed over a known period of time we can estimate the average execution time over that period. To implement this, the PLC program needs to be modified slightly to increment a counter for every scan cycle executed and report that count over whatever ICS protocol is already being used, such as Modbus. The monitoring device then simply measures the time elapsed and the count progression to calculate average scan cycle times, as in Figure 5.6.

When this method is again applied to the example programs on the MicroLogix 1100, in Figure 5.7, the programs are now easily separable. Note that similarly for the M221, even though the bottling and sequencer example appear very close, there is consistently more than a 3 μs difference in their execution times.

Even though the 1 ms resolution provided no information for classifying programs on the S7-1200, the improved measurement techniques now show clear differences in Figure 5.8.

For the sake of thoroughness, these improved measurement methods were also tested on the M221 and M241 PLCs. However, the surprising discovery was made that these PLCs

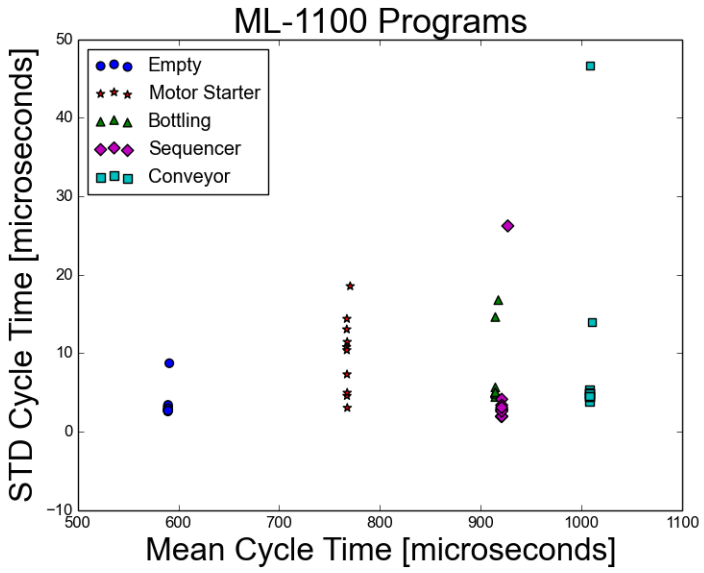


Figure 5.7: Improved Method: MicroLogix 1100 Scan Cycle Times

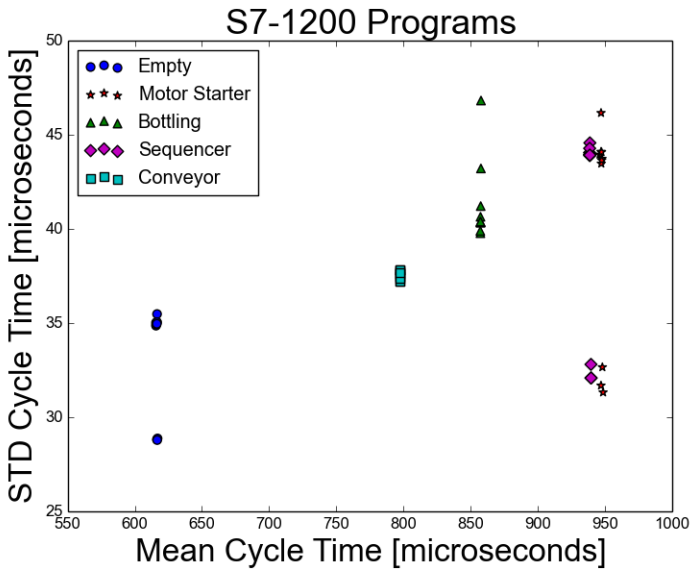


Figure 5.8: Improved Method: S7-1200 Scan Cycle Times

implement scan cycle execution differently than the other two vendors, and differently from a literal interpretation of their documentation. Both the M221 and M241 documentation claim that the ”freewheeling” mode of operation executes the scan cycles as fast as possible, similar to both the Allen Bradley and Siemens PLCs. When the count progression method was used to estimate the average execution times, the Modicon PLCs were instead found to execute at the fastest possible millisecond interval or even two-millisecond intervals. In other words, the system diagnostic information for the M221 would report scan cycle times in the hundreds of microseconds, but was found to actually be executing the control program every two milliseconds exactly. Similarly, when the M241 was reporting scan cycle times in the hundreds of microseconds, it was actually found to be executing the control program exactly every one millisecond. Presumably, the reasoning behind this discrepancy is to ensure the underlying system has enough CPU time to perform other critical functions. This limits the accuracy of the basic count progression method of scan cycle measurement, but modifications are proposed in Section 5.7 to work around this limitation.

For the remainder of this chapter, all measurements unless otherwise stated, were taken from the system diagnostics for the M221 and M241, and the improved count progression method for the MicroLogix 1100 and Siemens S7-1200.

5.5.4 Instructions

The previous section described obvious differences between different real-world PLC programs, but in order for the techniques to be useful, they must be able to detect minor changes in the program. As a first step in this direction, experiments were conducted to add increasing numbers of common instructions to empty programs in order to simply measure how much a single instruction impacts the cycle times. The most common instructions by far in PLC programming are the use of timer on delay blocks (TON), count up blocks (CTU), moving data into different variables (MOV), examine if closed (XIC) instructions,

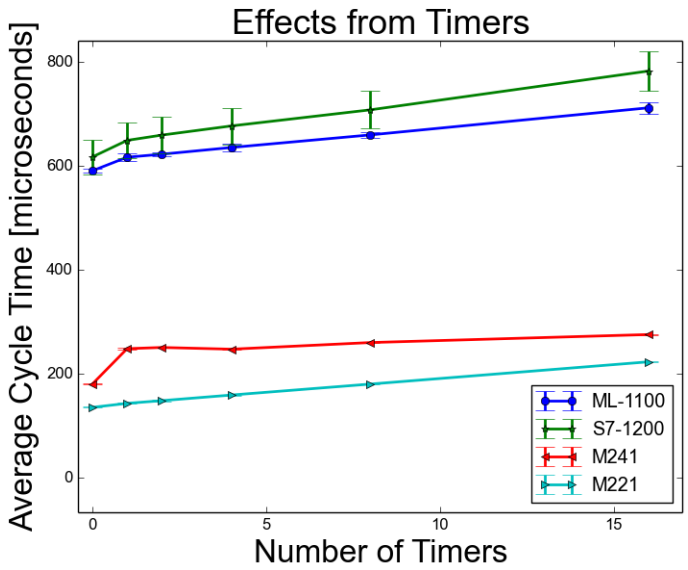


Figure 5.9: Effects of timer blocks on execution times

and output energize (OTE). Figures 5.9 through 5.11 illustrate the average execution times of programs with standard deviations represented with error bars.

For each PLC, every instruction takes a different amount of time, with MOV, XIC, and OTE instructions taking the least amount of time and timer and counter blocks taking more. Figures 5.9 through 5.11 illustrate average execution times for some of the instructions. For the sake of space, the remaining instructions may be found in Appendix 5.5.4. Execution times were calculated by taking the slope of the line representing the increase in total program execution time over the increase in number of instructions. These results are summarized in Table 5.3, and become useful for the white box modeling techniques proposed in Section 5.5.5. It is important to note that the XIC instructions represent IF checks on single bits, and that execution time of Boolean algebra operations on them (AND and OR) must also be calculated.

Instruction Execution Times

Figures 5.12 through 5.14 illustrate other common instruction execution times.

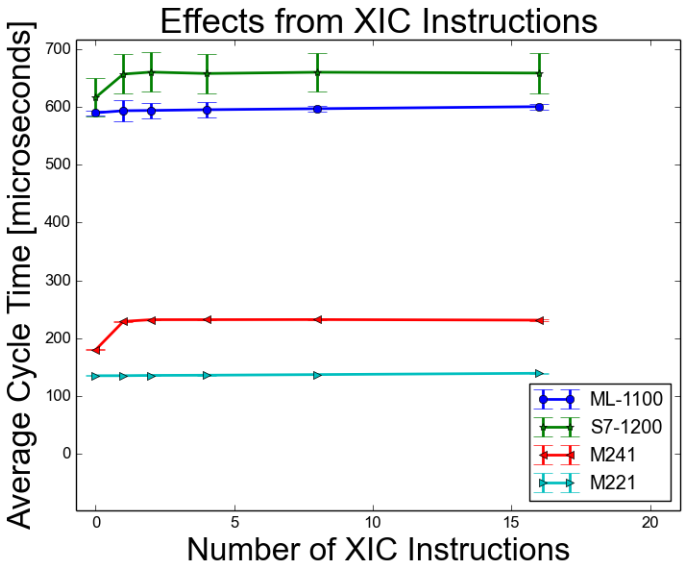


Figure 5.10: Effects of XIC instructions on execution times

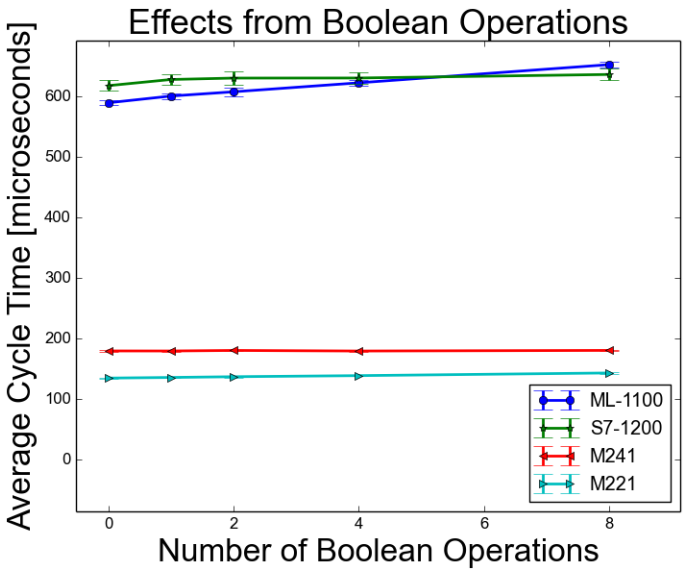


Figure 5.11: Effects of Boolean operations on execution times

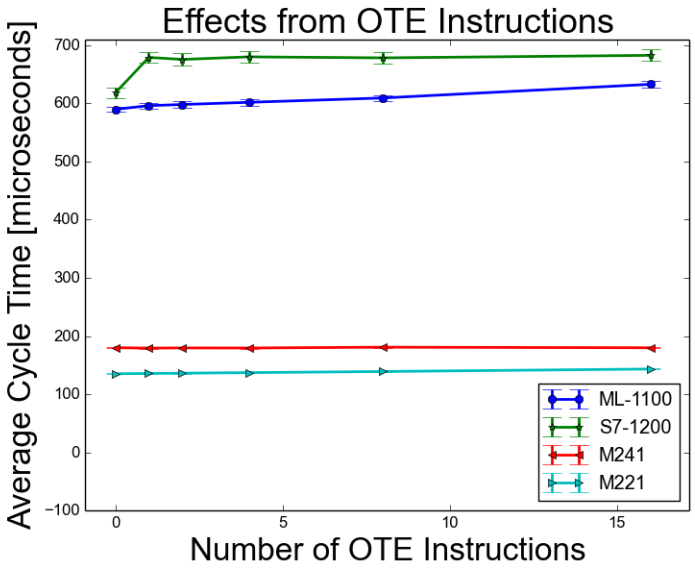


Figure 5.12: Effects of Output Energize (OTE) instruction on execution times

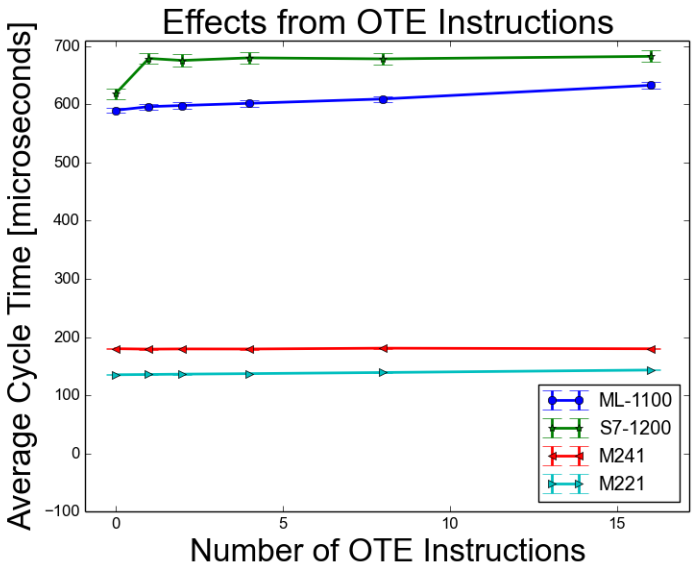


Figure 5.13: Effects of counter (CTU) instruction on execution times

Table 5.3: Average Instruction Execution Times [μs]

PLC	ML-1100	S7-1200	M221	M241
XIC	0.86	0.312	0.233	0.08
(AND/OR)	4.11	0.64	0.32	0.02
OTE	2.45	0.237	0.509	0.04
MOV	5.84	2.04	2.85	0.358
TON	6.33	8.89	5.32	1.81
CTU	7.06	9.89	7.91	1.24

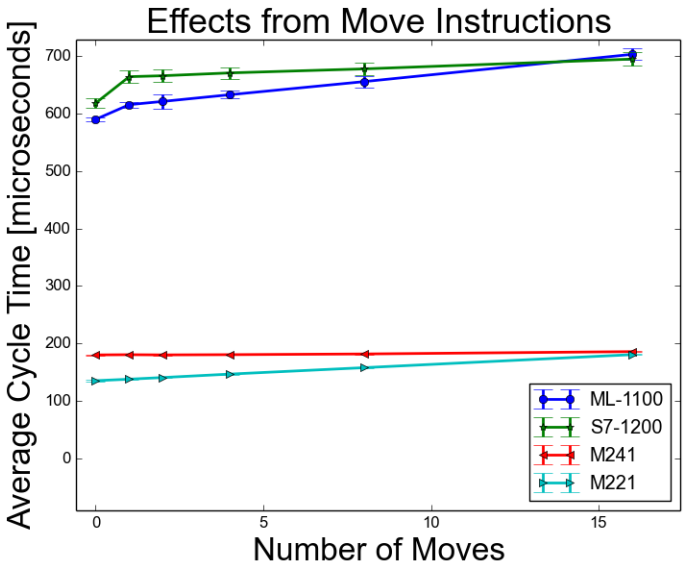


Figure 5.14: Effects of move operations on execution times

5.5.5 Program Branching

Most control program execution times are largely static in nature, essentially evaluating the same Boolean algebra expressions over and over again and not executing jump statements to significantly different execution paths. Slight variations occur depending on how quickly the Boolean expression is evaluated (e.g., anything ANDed with a 0 is 0), but over the hundreds of thousands of cycles that occur in a few seconds, these typically average out. All of the example programs found in this research exhibited static execution times and only used jump instructions to consistently and always execute the same subroutines.

However, there may exist cases where more complex control programs use jump statements to change the mode of operation, for example to a maintenance mode or when a fault

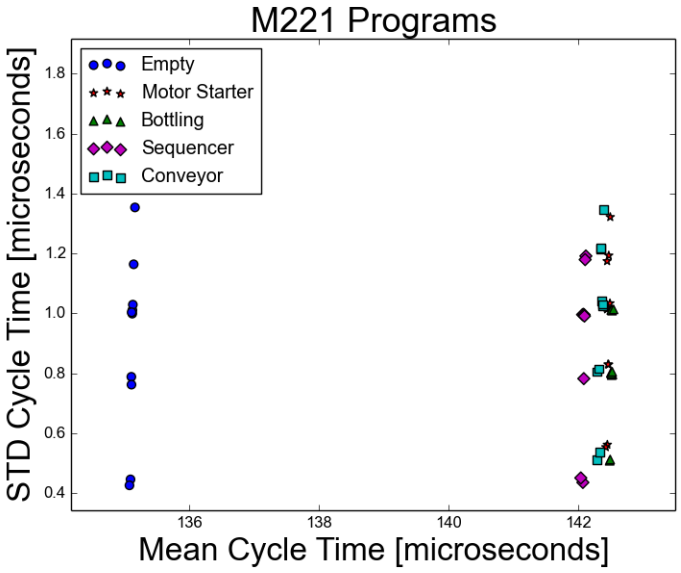


Figure 5.15: Execution times of fault modes on the M221

is detected, that significantly alters the execution times. To illustrate this, the four example programs used in this research were modified to have a branch path that simulated a fault scenario, where the original program is jumped over and outputs are simply set to a known ”safe state”. Note that with this modification, all example programs are now essentially executing the same instructions resulting in very close overlap, for example in Figures 5.15 and 5.16.

Different modes of execution, such as these fault modes, should ideally occur on a rare basis. However, given the approach described so far, any change detection algorithm will immediately begin alerting operators of a change in the program if the mode changes. Granted, this could be perfectly acceptable as the operators may in fact want to be alerted when the process has moved into an abnormal state. However, if it is a regularly occurring change in mode, as for maintenance, the change detection system would also have to train on what the maintenance mode executions look like, or know what the rare mode will look like before it observes it, which motivates the need for white box modeling approaches.

The previous techniques proposed in this chapter have all been based on real samples from real devices, making them black box techniques. However, in the unlikely scenario

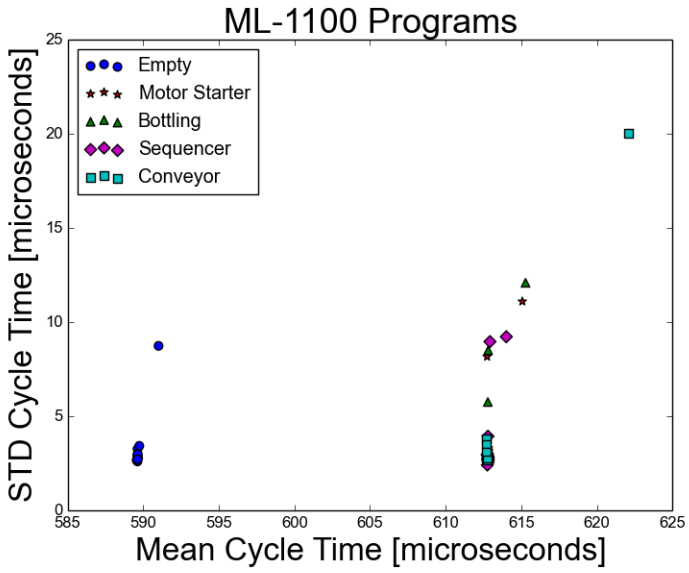


Figure 5.16: Execution times of fault modes on the ML-1100

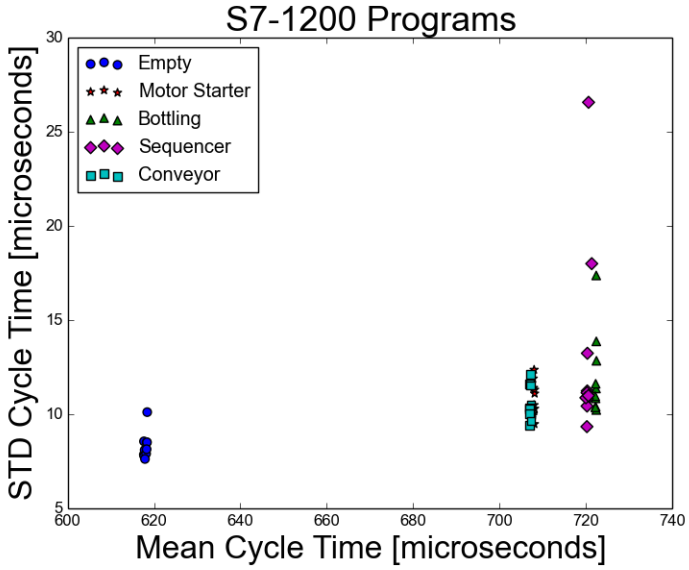


Figure 5.17: Execution times of fault modes on the S7-1200

that a control program has rare branching conditions that significantly alter the program execution times, those conditions may be so rare that it does not encounter them in the training period.

Fortunately, given access to the source code of the program and knowledge of the execution times of each instruction, the expected execution times of those rare branches can be constructed beforehand and the algorithm trained on those samples. This task is made even easier given that some vendors include in their documentation the average execution time of each instruction, as in the case of the MicroLogix 1100 [57]. If such information is not readily available, experiments as in Section 5.5.4 can be carried out to build a database of how long each instruction takes to execute.

To illustrate this, "fault" condition branching was implemented in the PLCs that consisted of simply a jump statement and setting ten outputs to a "safe" state. This equivalently is 10 OTE instructions and the jump statement. When the white box methods using the vendor supplied executions for the MicroLogix 1100 are applied to this condition, the ten OTE instructions are estimated to take 14.3 μs to 15 μs on top of the original average execution time of 590 μs for an empty program. This results in an estimated scan cycle range of 604.3 to 605 μs, about ten microseconds short of the actual observed execution times.

Using the more accurate measurements for the OTE obtained from Table 5.3, the white box methods estimate a mean execution time of 614 μs nearly identical to the times observed in Figure 5.16. The concept of white box modeling more complex execution paths is explored in more detail in Appendix 5.5.6, but the amount of uncertainty in knowing which instructions are executed resulted in lower accuracy.

5.5.6 White Box Modeling

To demonstrate how white box modeling might work for more complex program branches, consider the Motor Starter program for the MicroLogix 1100. Table 5.4 lists the total number of instructions in the program and their corresponding average execution times

from the vendor documentation. Note that XIO is simply the negated form of XIC, JSR is the ”jump to subroutine” instruction, and END is simply returning from each subroutine. When the totals are added up, the estimated range of control program execution time was between 141.88 and 159.52 μs. Given the average execution time of an empty program was approximately 590 μs, the white box model estimate of the average scan cycle time for the motor starter program comes to a range of 731.88 to 749.52 μs. Note that this is close, but a slight underestimate of the observed execution time around 760 μs.

Table 5.4: White Box Model for Motor Starter on ML-1100

Instruction	Quantity	Min Time [μs]	Max Time [μs]
XIC	54	1.45	1.5
XIO	12	1.5	1.5
OTE	16	1.43	1.5
TON	8	2.59	4.06
JSR	2	0.84	1.87
END	3	0.1	0.1
Total		141.88	159.52

This underestimate can be attributed to the inaccurate execution times from the manual discovered earlier in Section 5.5.4. Instead of thinking of XIO and XIC instructions as single instructions themselves, one must also consider whether they are being executed in series or parallel (i.e. being ANDed or ORed). If several XIC instructions are in series and the first one is evaluated as True, then there is no need to evaluate the rest. Likewise if XIC instructions are in parallel and the first is evaluated as False, the remaining XIC instructions do not need to be executed. Therefore, we must estimate how many of the additional XIC instructions result in AND or OR operations that immediately return an answer for the entire statement. One option would be to make the naive assumption that for every additional XIC instruction, there is a 50% chance that it will immediately return an answer. However, since the primary purpose of the change detection algorithm is to detect positive change (i.e. *added* instructions), the white box model should err on the side of overestimating the mean to avoid a large number of false positives. Choosing a

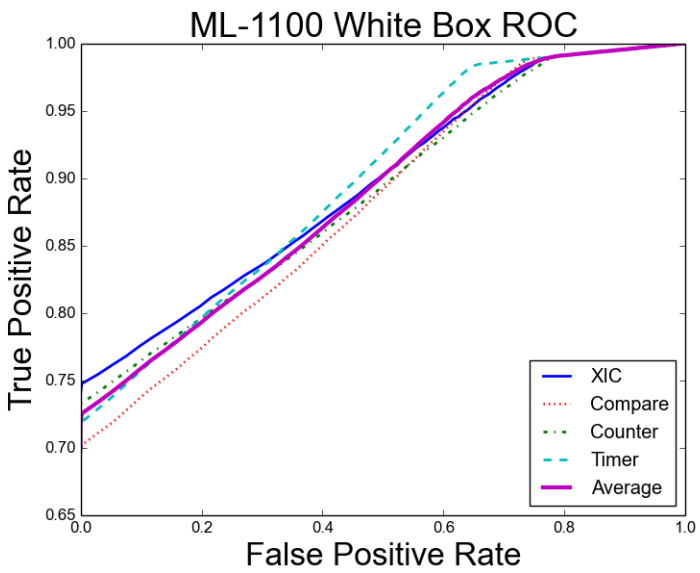


Figure 5.18: White Box Model ROC for ML-1100

conservative estimate that 66% of the Boolean operations are executed results in Equation 5.1, where t_{XIC} represents the execution time of all XIC instructions in that rung, n is the number of execution times, and B_t is the execution time for Boolean operations. Using this equation and the more accurate measurements of execution times from Table 5.3, again we recreate the Motor Starter program for the MicroLogix 1100.

$$t_{XIC} = \begin{cases} XIC_t & n = 1 \\ XIC_t + 0.66^{n-1} * (B_t) * (n-1) & n > 1 \end{cases} \tag{5.1}$$

Building a white box model using this equation and the values in Table 5.3 for all other calculations, gives a white box estimated mean of 763.5 μs, close to the observed mean time of 760 μs but as expected, slightly over. When this estimated time is fed into the change detection algorithm, the classification results are slightly lower, illustrated in Figure 5.18, as 70% true positive rate with zero false positives.

5.6 Change Detection

5.6.1 Theory

The end goal of the proposed change detection techniques is to make it infeasibly difficult for an adversary to modify the original control program to achieve stealthy goals. In order to remain stealthy, he must not only evade any network intrusion detection tools already in place, but he must also refrain from causing any significant immediate changes to the physical process, or the operators will immediately know something is wrong. The easiest way to do this would be to add a logic bomb on top of the original program that only sabotages the process on rare occasions, or until some signal is given to coordinate an attack across multiple PLCs. Therefore at the absolute minimum, an attacker must add some form of trigger or logic check to the original code to implement his logic bomb. In the case of PLC programming, these triggers could take the form of a timer, counter, comparison statement, or a simple examine if closed (XIC) instruction. If the proposed scan cycle based change detection techniques can detect a single addition of any of these instruction, they can therefore detect the addition of any form of logic bomb.

While there are many options for change detection algorithms to apply to this scenario, the basic cumulative sum (CUSUM) method from 1954 [58] was chosen to illustrate performance. More sophisticated algorithms exist and could potentially perform even better than the impressive results achieved here. The CUSUM method intuitively operates on the idea of cumulatively adding the difference between a variable and its known mean over time. If that cumulative sum ever passes a certain threshold, then the decision is made that a change has occurred. More formally, it is defined according to Equation 5.2, where S_n represents the cumulative sum value at sample n, x_n represents the value being monitored

at sample number n, and w_n is some weight, usually the mean value of x.

$$\begin{aligned} S_0 &= 0 \\ S_{n+1} &= max(0, S_n + x_n - w_n) \end{aligned} \tag{5.2}$$

CUSUM in the form of Equation 5.2 is designed to detect positive change in a variable, and so only this form of CUSUM was used in this research, since any addition of code to add a logic bomb will likely cause a positive increase in scan cycle times. Negative changes can also be detected using the min function instead of max in the equation and alerting when S_n crosses below the negative threshold. Application of this form of CUSUM could be used to detect malicious removal of instructions and shortening of the scan cycle time, illustrated briefly in Appendix 5.6.3. The negative change detection form of CUSUM is also demonstrated in Section 5.7 with the application of proof-of-work functions.

5.6.2 Results

To test the performance of the CUSUM change detection algorithm applied to scan cycle times, training data was taken from half of the original samples for each of the example programs. The mean execution times were measured over this training set and used as the w_n variable in Equation 5.2. The testing data consisted of the remaining half of the original example programs as well as 10,000 samples each for four different modifications of the original program. These modifications included all possible methods for the minimum requirements for an attacker to insert a logic bomb into the code: adding a single timer, a counter, a comparison statement, or an examine if closed (XIC) instruction. The false positive rate was defined as the ratio of the number of times a sample was incorrectly classified as a change over the total number of samples from the unmodified test set. The true positive rate was defined as the ratio of the number of samples correctly classified as changes over the total number of samples from the modified test sets.

With these definitions, receiver operating characteristic (ROC) curves were plotted for

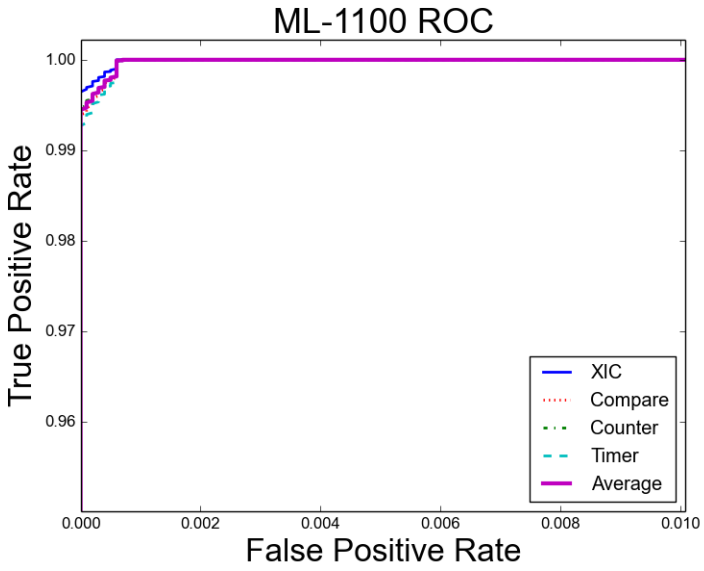
each PLC, comparing the trade-off between false positive rates and true positive rates. Figure 5.19a illustrates classification performance for the MicroLogix 1100. Due to the processor being so slow and each additional instruction adding so much time, detection rates were extremely high with over 99% true positive rates at zero false positives. The CUSUM change detection algorithm is sequential, in that it operates on every sample that arrives, which means that true positive rates are not the only metric that should be calculated. Instead, the average time to detection is used to measure how quickly the algorithm can detect the change. For these experiments, average detection times for the given false positive rates were calculated by choosing a random place to start in the test data of modified programs and measuring the number of samples the algorithm took to first alert on the change. With false positive rates of less than 0.5%, the average detection time for the MicroLogix 1100 was in the tens of samples, which with the sampling rate of 10Hz used here corresponds to alerting in a few seconds.

The other PLCs are built with faster processors, so the minuscule change of adding a single XIC instruction proved harder to detect. However, the corresponding average detection times at a zero false positive rate were all within minutes of the program changing.

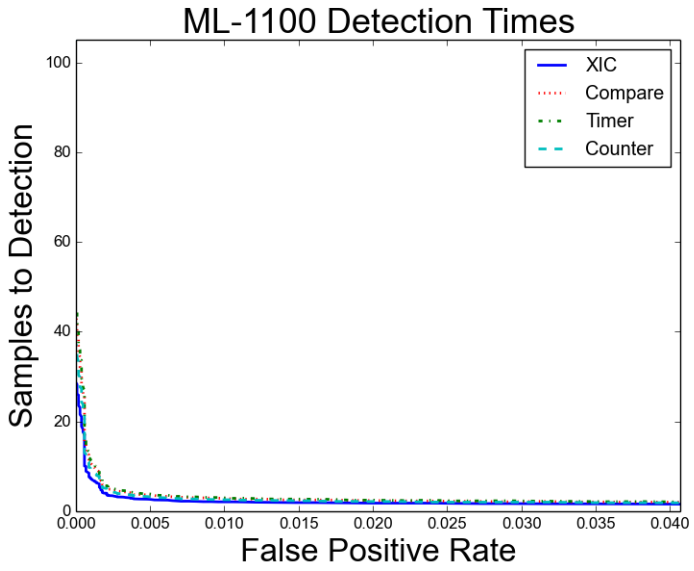
Although positive change detection was the primary threat model addressed here in order to detect the addition of logic bombs, negative change detection was also considered for the case where an attacker is removing instructions from the program. This is explored in more detail in Appendix 5.6.3 and was found to achieve similar detection accuracies.

5.6.3 Negative Change Detection

Negative change detection using the CUSUM algorithm is achieved by simply using the ”min” function instead of the ”max” function, and alerting when the value of S falls below

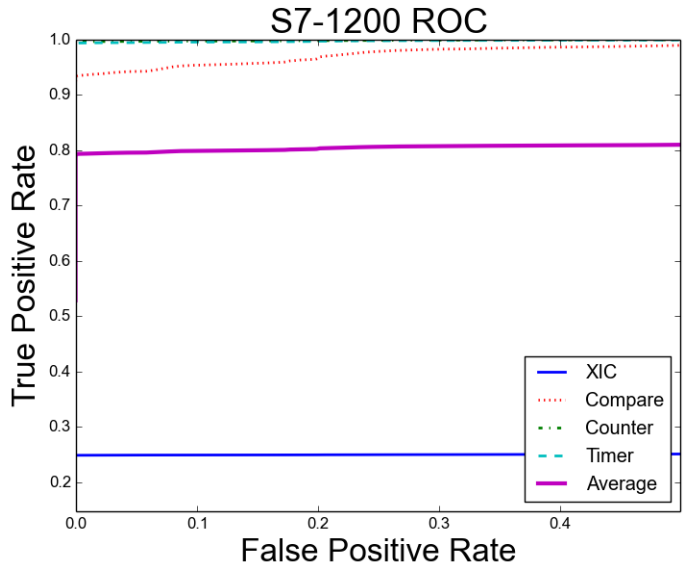


(a) ROC Curve for MicroLogix 1100



(b) Detection Times for MicroLogix 1100

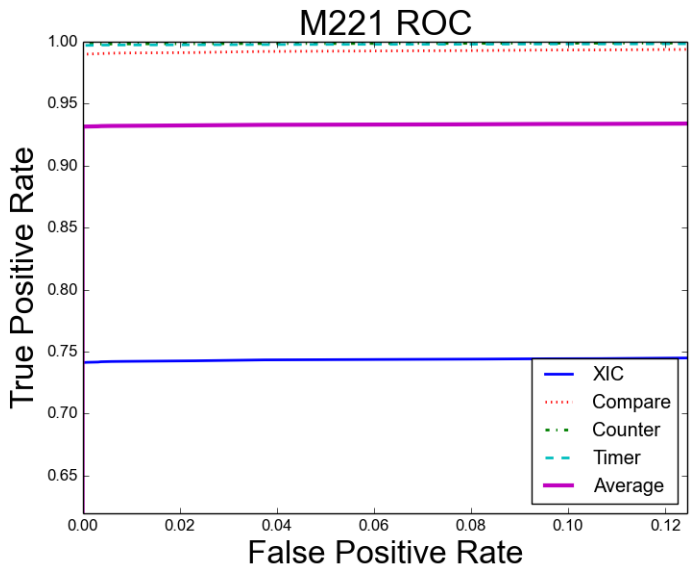
Figure 5.19: MicroLogix 1100 Performance



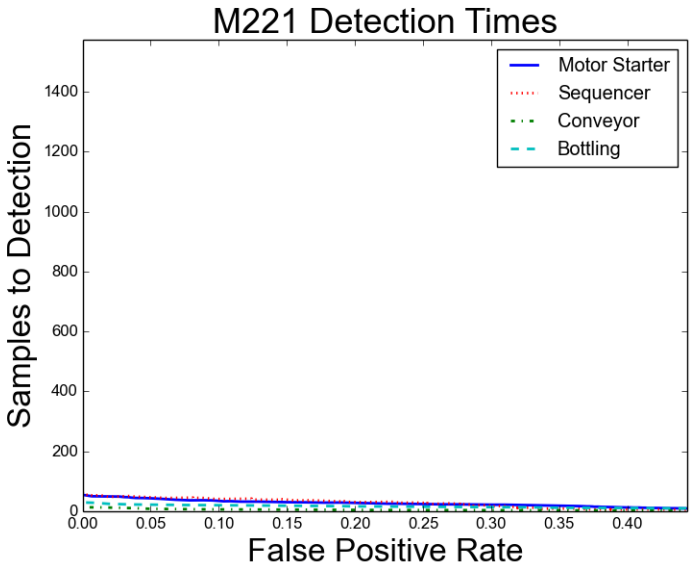
(a) ROC Curve for S7-1200

(b) Detection Times for S7-1200

Figure 5.20: S7-1200 Performance

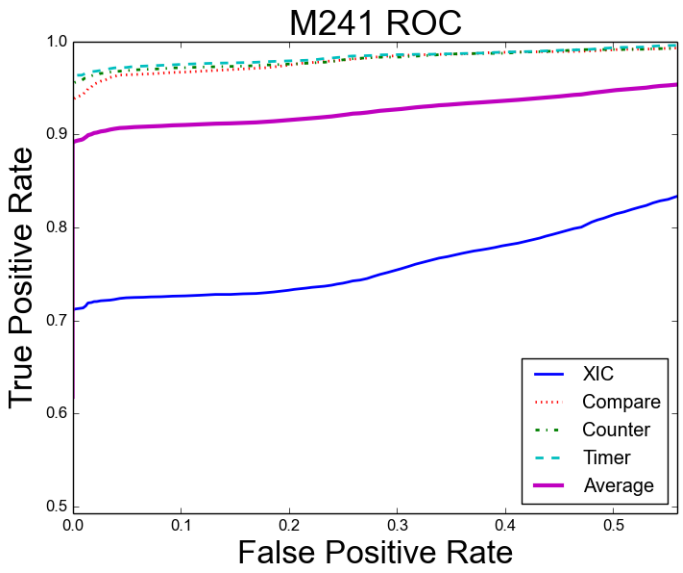


(a) ROC Curve for M221

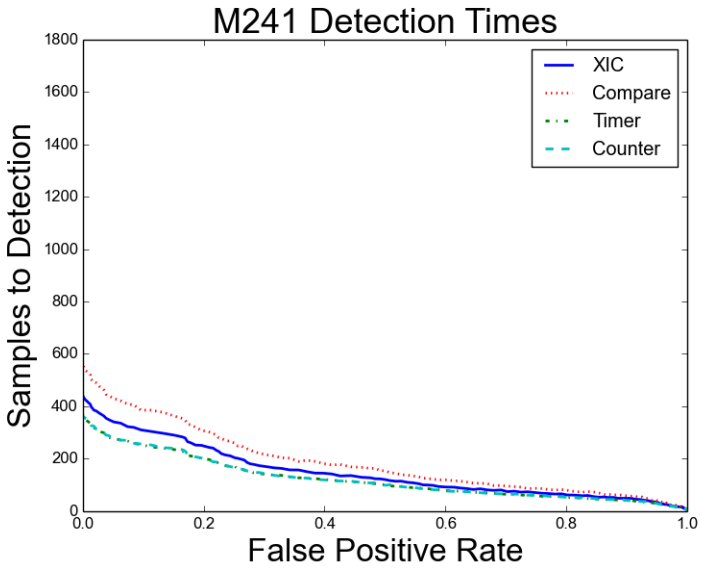


(b) Detection Times for M221

Figure 5.21: M221 Performance



(a) ROC Curve for M241



(b) Detection Times for M241

Figure 5.22: M241 Performance

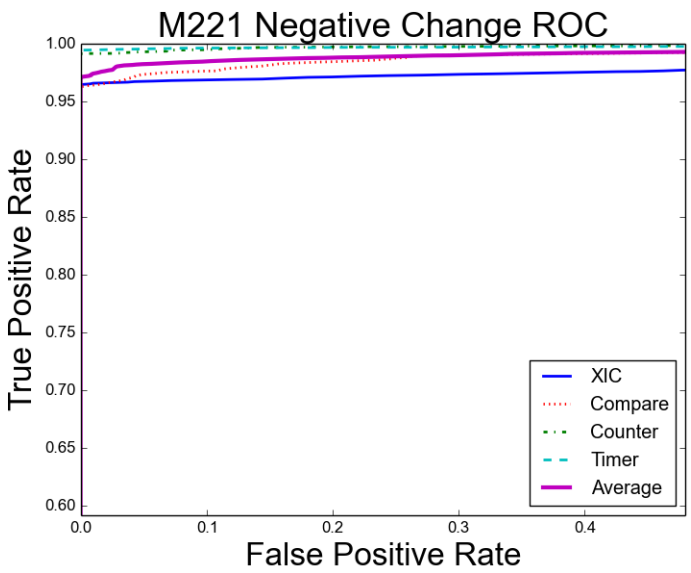


Figure 5.23: Negative Change Detection for M221

the negative threshold, as described in Equation 5.3.

$$S_0 = 0$$
$$S_{n+1} = min(0, S_n + x_n - w_n) \tag{5.3}$$

To test negative change detection, the baseline was first trained on half of the modified program data (the versions with added instructions) as ”normal” and then tested on the original versions. Since this is virtually testing the same changes, performance is essentially the same, as illustrated in Figure 5.23.

5.7 Robust Change Detection

The previous techniques are able to detect low and medium skilled attackers, but they fail to address the scenario where an attacker is modifying the scan cycle counts being used for the improved accuracy measurements. To address this level of attacker, proof-of-work functions

5.7.1 Proof-of-Work Theory

Traditionally, proof-of-work (POW) functions have been used in scenarios to prove a node has committed a minimum number of clock cycles before dedicating any system resources to handling their connection, in order to mitigate denial of service attacks. They achieve this by using a special class of functions that are computationally expensive to solve, but orders of magnitude easier to verify.

For example, consider the discrete logarithm problem which is the foundation of most public key cryptography algorithms used today [59]. The underlying theory is that given proper choices of a large prime p and a g that is a primitive root modulo p, there is no efficient method to solve for the integer k in the equation $g^k mod\, p = b$ where b is a member of the multiplicative group of integers modulo p or $b \in Z_p^*$. The most efficient algorithm proposed to date to solve this problem has running time that is exponential in half the number of digits in the size of the group. However, verification of answers to the discrete log problem can efficiently be achieved using exponentiation by squaring, which requires $\lfloor \log_2 n \rfloor$ multiplications and at most $\lfloor \log_2 n \rfloor$ multiplications, where n is the exponent.

By taking advantage of the severe resource limitations on PLCs (kilobytes of RAM and Megahertz processors) and their deterministic real-time operating systems, POW functions can be used to provide an upper bound guarantee that the PLC is not executing additional instructions in the background. This is accomplished with the combination of two applications of POW functions:

- Measuring scan cycle count progress
- Proof of minimum scan cycle time

The first application of POW functions for PLCs mirrors the scan cycle count method explained in Section 5.5.3. Recall that the method is based on the fact that even the smallest consistent addition of instructions in the control program will affect the scan cycle slope and eventually be detected if monitored for long enough. Unfortunately, that simple approach

suffers from the weakness that an intelligent attacker can easily modify the addition of scan cycle counts to match the original slope since an ”addition” instruction takes the same amount of CPU cycles whether it is adding a one, or one hundred. An attacker however cannot skip steps in solving the POW function and simply add one-hundred to the counts, instead he must actually execute the one-hundred steps in the POW function while still keeping the execution time of the program consistent with the original slope. This is only possible if he first removes instructions from the original program, thus increasing the chances that he will cause some form of immediate change in the physical process that the operators will notice.

The second application of POW functions covers the scenarios discovered in Section 5.5.3, where the control program is not executed as fast as possible, and actually is only executed at fixed intervals. The main idea behind this second layer of POW functions is essentially to fill up the idle time of a scan cycle with steps of the POW function up to a desired maximum scan cycle time. Again, an attacker cannot skip any steps in the function, but must actually execute them in order to return the correct answers to the verifier. The verifier can then regularly poll the PLC for the supposed number of scan cycles executed and the current progress on both POW functions. If the scan cycle time estimates that the verifier calculates from those polls do not match the known signature, or if the POW verification fails, he knows that the PLC has been modified.

To demonstrate how POW functions may be applied for intrusion detection in resource-starved real-time systems, the previous discrete logarithm problem was chosen. However, one difficulty in applying POW functions to PLCs is that the severe resource limitations must be taken into account when designing the POW function. Care must be taken that the computational load of solving the POW problem does not significantly effect the execution of the control program. Furthermore, some of the PLCs can only perform the mathematical operations, such as the modulus operator, on 16-bit numbers, so the parameters of the discrete log problem must be chosen carefully. This latter condition must also be balanced

with the security requirement that the size of the group of integers modulo p must be large enough that it is infeasible for an attacker to reliably guess the pseudo-random results obtained in the steps of solving the POW function.

For the purposes of this research, it was decided that a minimum group size of 1000 would provide a strong enough guarantee that the results would not be consistently guessed correctly. This effectively means that at least 10 bits would be required to store the value of p. To find the upper limit on the size of q, we consider the requirement for it to be infeasible for an attacker to pre-compute the results and store them on the PLC without removing the original program. Given that the application memory available on three of the four PLCs used in this research was less than 1MB, it was decided to design the POW function as to require at least 1MB of memory to store all possible precomputed results. Using an upper limit of 12 bits to store q provides a range of choices for q including 396 primes, each providing a group size on average of 2509. Assuming 2 bytes are required to store each step in the POW function comes to a total storage requirement of $M = 396 * 2509 * 2 = 1.99MB$, assuming the verifier properly cycles through all POW variables. Note that more powerful PLCs with more memory are also likely to have larger processors capable of operating on 32-bit numbers, allowing this parameter to be appropriately modified. Finally, with the assumption that a maximum of 12 bits are used to store q, a maximum limit of 4 bits is left for g.

To illustrate the steps in the POW function, consider the discrete log problem where $q = 3779$ and we are using the primitive root $g = 6$ for the generator. For each scan cycle, the PLC multiplies the current POW value by 6, divides by 3779, stores the remainder as the current progress, and keeps track of how many scan cycles have been executed, which is equivalently k from the the discrete log equation. Now at any point in time, the verifier can ask how many scan cycles have been executed (k) and what is the corresponding POW value, and using exponentiation by squaring quickly confirm that the two match.

5.7.2 Results

For the MicroLogix 1100, the scan cycles are executed as fast as possible, so only the first method of POW functions was necessary. As a reminder, this involves the PLC taking one step towards solving a POW for every scan cycle. When this method was applied, again the detection accuracy was high, with Figures 5.24a and 5.24b showing a true positive rate of 98.5% and an average detection time of 25 samples (2.5 seconds) at zero false positives for the test set.

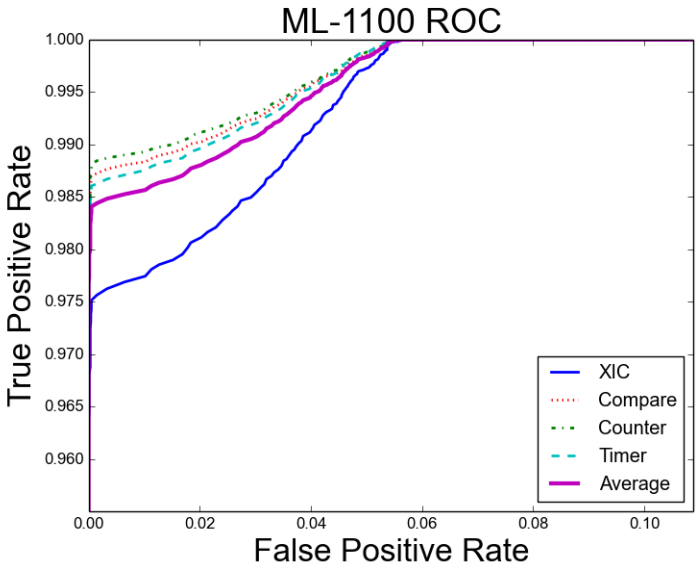
Where previously with the Schneider Modicon PLCs the true execution times were only on millisecond boundaries, we can now use the POW functions to verify that a certain amount of work was completed each scan cycle, which means that we can verify that no additional instructions were being executed in the background. Plotting the mean number of POW steps completed per scan cycle time reveals how much idle time there was between the true execution time and the millisecond boundary, resulting in the distributions in Figures 5.25 and 5.26 being mirrored versions of the execution times observed in Figure 5.3 and 5.4.

When the negative change detection form of CUSUM is applied to the number of POW steps taken per scan cycle, we can effectively

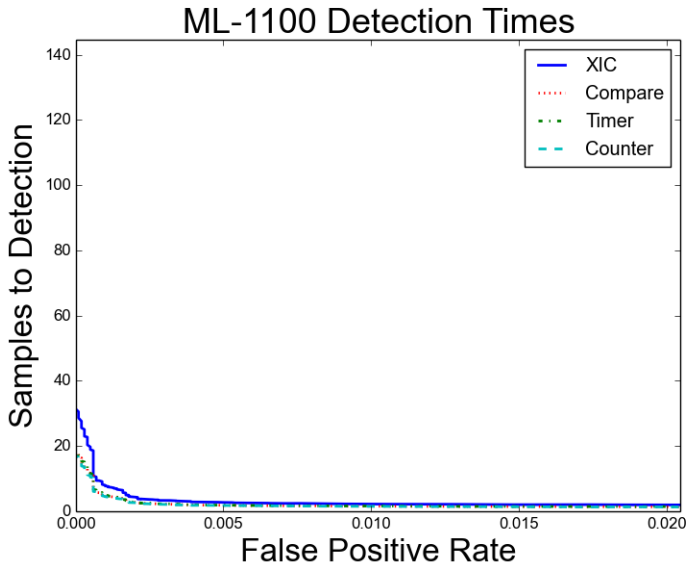
5.8 Discussion

5.8.1 Performance

Detection performance both in terms of ROC curves and average detection times, were found to be acceptable for all PLCs and example programs. Timers, counters, and compare statements were all detected with high accuracies above 90% true positive rates at less than 1% false positives. XIC instructions proved more difficult, but still were detectable at rates above 75%. This means that an attacker attempting to add *any* form of logic bomb must remove instructions from the original program while not causing immediate disruption to



(a) ML-1100 Proof of Work ROC



(b) ML-1100 Proof of Work Detection Time

Figure 5.24: ML-1100 POW Performance

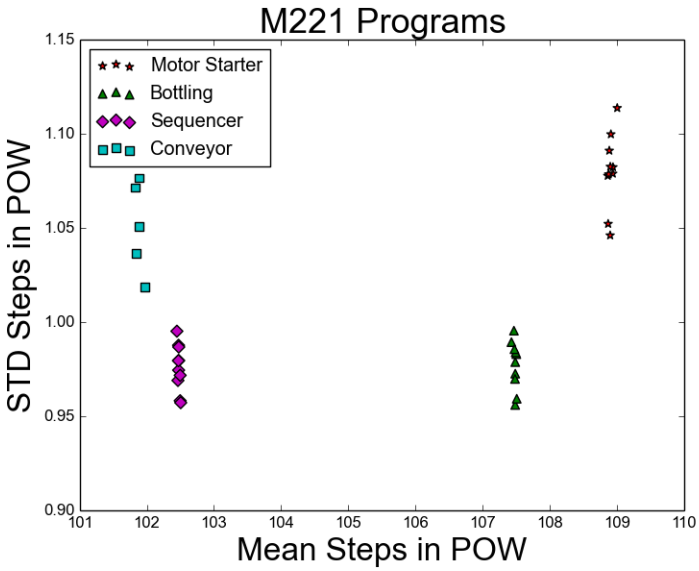


Figure 5.25: Mirrored Distribution of M221 Using POW Steps

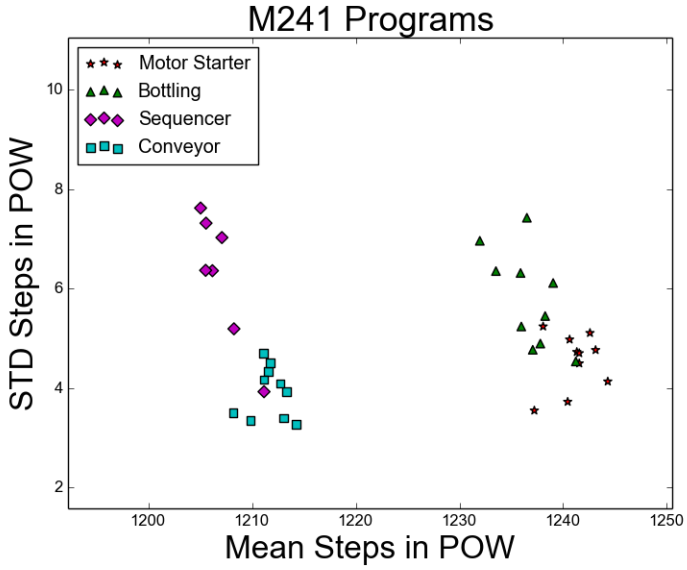
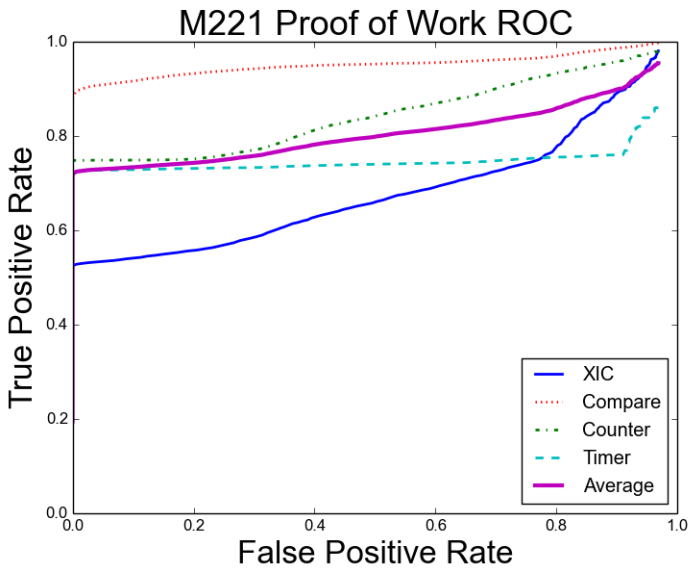
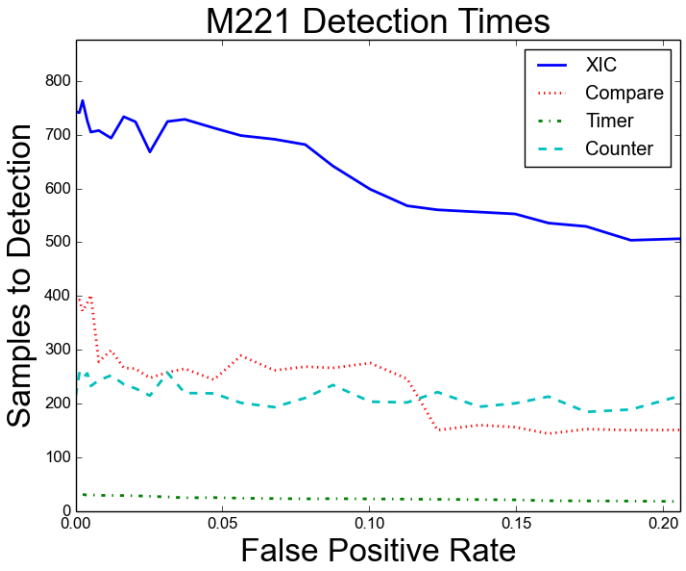


Figure 5.26: Mirrored Distribution of M241 Using POW Steps



(a) M221 Proof of Work ROC



(b) M221 Proof of Work Detection Time

Figure 5.27: M221 POW Performance

the physical process in order to remain stealthy. Average detection times at false positive rates less than 1% ranged from seconds to no more than minutes.

Possibly the most significant aspect of this accurate performance is the fact that this method performs just the same regardless of whether the PLC was reprogrammed from the network (which could be detected using other means), but *also from potential insiders with direct physical access to local programming port*.

5.8.2 Attacks

The threat model addressed in this research assumes that an attacker is attempting to maliciously modify an existing PLC program at the application layer while remaining stealthy. As a reminder, this mirrors the only real-world attack on PLCs to date and is most feasibly achieved if the attacker inserts a logic bomb into an existing control program. We now consider how the proposed change detection techniques perform against a variety of attacks previously designed against software attestation techniques.

Code Optimization

One of the first papers describing attacks against the SWATT software attestation protocol suggested that the security of the techniques rely on two important assumptions: 1) That the SWATT algorithm is using the most efficient implementation of the checksum and 2) That they were correctly assuming the most efficient method of attack [60]. The authors then implement a more efficient attack than the SWATT algorithm assumed, but still admit that the SWATT verifier could detect their attack.

While the second assumption directly applies to the methods proposed here, the first assumption requires some rephrasing to fit this setup. In the context of the application layer PLC program, the security of the proposed research depends on the assumption that a) the original program is already the most efficient implementation, and b) the POW function used is the most efficient implementation. If an attacker is able to optimize either the

original program or the POW function used, he may be able to save enough time to insert some malicious instructions.

Regarding assumption 1.a), an attacker attempting to optimize a PLC program is going to encounter many obstacles. First recall that the attacker model here is one where the adversary is not modifying the firmware, but simply loading a new application layer program, written in ladder logic for example. The attacker can only modify this ladder logic program and does not have the ability to directly control what CPU level instructions the program gets compiled to, making it significantly harder to automatically find optimizations. Second, PLC programs are so singular in purpose they are already very simple, leaving little room for optimization. The example programs here only consisted of around 500 total instructions each, with a fraction of those being executed every scan cycle. The chances of an adversary being able to find an application layer optimization in this small amount of code and replace it with malicious instructions that achieve his goal while also taking the same amount of time to execute is very small.

Many of the same arguments can be made for assumption 1.b), that the implementation of the POW function is already the most efficient. Again, the attacker can only modify application layer instructions performing the POW, not the underlying CPU instructions. In the research proposed here using the discrete logarithm problem, each step in the POW consisted simply of one multiplication instruction and one division instruction, leaving no room for optimization. More efficient methods exist for calculating discrete logarithms, but most require large memory space for precomputing results. Furthermore, the goal of these more efficient methods is to solve a single POW problem, without showing all of the steps taken to get the solution. In the methods proposed here, progress on the POW problem is constantly being monitored, meaning any theoretical attack on the POW function must be more efficient per step (i.e., the multiplication and division instructions), or must be so much more efficient that it is capable of solving POW functions at the rate progress is being polled for.

Finally, assumption 2) assumes that the methods proposed here address the most efficient method of attack. In this case we considered the addition of a single XIC instruction in the main program loop and were able to reliably detect it. There is no way an adversary can be more efficient than a single XIC instruction unless he inserts the instruction into an already rarely executed branch of the program. While the techniques proposed here would likely not be able to detect additions in a rare branch, this severely limits the kind of goals an attacker could achieve.

Proxy Attack

Now consider the case of a ”proxy attack” defined in previous work on remote attestation[28]. In this attack, the device under test communicates with a faster proxy to calculate the correct answer to the verification challenge instead of calculating the answer itself. Unfortunately there are no hard theoretical guarantees that this isn't possible against the methods proposed here. However, there are a number practical reasons why it is infeasible. Most importantly, the proposed use-case for the techniques would be part of an intrusion detection system monitoring the ICS network. Therefore, if the PLC reaches out to a proxy on the network for assistance, the attacker must also evade traditional ICS IDS techniques already in place that look at the static topology and relatively static bandwidth usage. PLCs typically only communicate with one or two other devices on the network and the communication consists almost entirely of polling for data, with much more infrequent ”write” commands to the PLCs. In order for a proxy to accurately recreate the PLC's scan cycle progression, it must update and write new values to the PLC *at least* as fast as the rate at which the PLC is being polled for POW progress. In a typical network where ”write” frequencies are on the order of minutes or hours, a sudden and consistent increase to several ”writes” per second would immediately raise alarms.

Other

5.8.3 System Impact

In order for any intrusion detection technique to be feasible, it must not significantly interfere with the existing devices or network. The proposed approaches would add a small amount of additional network traffic and in the case of the count progression measurement and the POW functions, a small amount of code to execute on the PLC.

Regarding network traffic, PLCs are already typically polled for multiple process measurements several times per second. To the process control engineers on the plant floor, the scan cycle times being polled for would just be another measurement among the dozens they are already taking from the PLC. Furthermore, the system diagnostic connections designed by the vendors of the PLCs themselves poll the PLCs up to ten times per second for information including scan cycle times, device name and model, and the current values of all variables in the control program without any detrimental effects to the PLC or process control network.

For the POW functions being added to the PLCs, the POW problem can be designed to guarantee some maximum amount of time is never exceeded on the POW function and thus some maximum amount of overhead is added. In the examples illustrated in this chapter, the POW function was designed to simply extend the scan cycle to the next highest millisecond boundary, guaranteeing that never more than a millisecond was added. Given that scan cycle watchdog timer options are usually in the hundreds of milliseconds, and that programs and devices are typically over-provisioned for safety, this is an acceptable amount of overhead.

5.9 Conclusions and Future Work

PLCs are the vital link between the cyber and physical worlds, but unfortunately they are still being sold insecure by design. This is unlikely to change anytime soon so in order

to protect ICS networks from potential life-threatening cyber attacks, scalable and immediately deployable techniques are required to protect the PLCs. To address this, change detection algorithms were applied to monitor the control program execution time of the real-time deterministic PLCs and alert on changes. At false positive rates less than 1%, average true positive rates of greater than 90% were achieved with detection times on the order of seconds or minutes. The basic techniques were extended to include more accurate measurements of the scan cycle times and white box modeling to predict execution times of rarely executed program branches. Finally proof-of-work functions were leveraged against the resource-starved PLCs to provide more robust guarantees that no additional code is being executed by the PLC. High accuracy performance and the difficult, but not impossible, attacks suggest the methods could add significant value as input into an ICS network anomaly detection system. Future work in this direction will investigate stronger adversary models attacking the underlying firmware, and will explore more accurate methods for generating white box model estimates of execution times.

CHAPTER 6
SUMMARY OF CONCLUSIONS

Despite the alarm constantly being raised about the insecurity of ICS networks for over 10 years now, a significant amount of work remains to be done primarily due to three main reasons. First, until know, little published data existed to educate researchers on what real-world ICS networks even look like. Without such knowledge accurate simulations and intrusion detection algorithms are very difficult to design. Second, attacks against ICS networks are different from IT networks both in their methods and goals. An entirely new class of attack based on false data injection is of primary concern for ICS networks, and the goals typically focus on causing physical damage rather than stealing information. Finally, standard IT approaches to network security do not translate well into the ICS environment. The sensitive nature of ICS devices limits patch frequency and makes any kind of active defense extremely undesirable, requiring strong passive techniques to provide security.

To address these many problems, fingerprinting techniques operating over ICS networks were proposed to identify various aspects of the system including software, hardware, and physical devices for use in intrusion detection systems. First, a detailed traffic characterization was performed on several power substation networks to examine whether common stated assumptions about ICS networks were true, and to guide the development of the fingerprinting techniques. One of the most interesting observations was that TCP round trip times for the resource-starved embedded devices were heavily clustered based on device type no matter how large the physical distance between them, suggesting they were largely based on processing time. This insight led to the development of cross-layer response time fingerprinting to passively identify device types based on the processing time between TCP level acknowledgments and application layer responses. Device type classification accuracy using standard machine learning techniques reached as high as 99% on

real-world substation traffic. To complement these techniques by addressing a different aspect of ICS networks, methods were developed to fingerprint the physical devices of the ICS. Previous work on physical fingerprinting was extended to improve relay classification from 92% to 100% and extend the scope of the methods to valves, motors, and pumps. Perfect classification accuracy was attained for pumps and valves, with motor classification reaching 88% true positive rate. Building on the idea behind the cross-layer response time methods, techniques were then explored that expand the scope to general programmable logic controllers by generating program fingerprints from the execution times of control programs. True positive rates for detecting program changes were around 90% with false positive rates under 1% and detection times on the order of minutes. The security of this technique was enhanced by the addition of proof-of-work functions to provide a provable upper bound guarantee that no additional instructions are being executed in the program. When combined, all of the proposed fingerprinting techniques provide the passive security, accuracy, and scalability necessary to form a robust, comprehensive ICS network anomaly detection system.

REFERENCES

[1] J.-W. Wang and L.-L. Rong, “”cascade-based attack vulnerability on the us power grid ”,” *Safety Science*, vol. 47, no. 10, pp. 1332 –1336, 2009.

[2] M. Abrams and J. Weiss, *Malicious control system cyber security attack case study-maroochy water services, australia*, http://csrc.nist.gov/groups/SMA/fisma/ics/documents/Maroochy-Water-Services-Case-Study_report.pdf, 2008.

[3] ICS-CERT, *Icsa-16-070-01*, https://ics-cert.us-cert.gov/advisories/ICSA-16-070-01, 2016.

[4] D. Formby, P. Srinivasan, A. Leonard, J. Rogers, and R. Beyah, “Who’s in control of your control system? device fingerprinting for industrial control system networks,” in *2016 Symposium on Network and Distributed System Security (NDSS’16)*, San Diego, California, 2016.

[5] *Nmap - free security scanner for network exploration & security audits*, http://nmap.org/, Accessed 2015-11-23.

[6] M. Zalewski, *P0f v3*, http://lcamtuf.coredump.cx/p0f3/, Accessed 2015-11-23.

[7] V. Paxson, “End-to-end internet packet dynamics,” *Networking, IEEE/ACM Transactions on*, vol. 7, no. 3, pp. 277–292, 1999.

[8] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and S. Diot, “Packet-level traffic measurements from the sprint ip backbone,” *Network, IEEE*, vol. 17, no. 6, pp. 6–16, 2003.

[9] Q. Shan, I. Glover, P. Moore, I. Portugues, R. Watson, and R. Rutherford, “Performance of zigbee in electricity supply substations,” in *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, 2007, pp. 3871–3874.

[10] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, “A first look at cellular machine-to-machine traffic: large scale measurement and characterization,” in *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS ’12, London, England, UK: ACM, 2012, pp. 65–76, ISBN: 978-1-4503-1097-0.

[11] R. Barbosa, R. Sadre, and A. Pras, "A first look into scada network traffic," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, 2012, pp. 518–521.

[12] S. S. Jung, D. Formby, C. Day, and R. Beyah, "A first look at machine-to-machine power grid network traffic," in *Smart Grid Communications (SmartGridComm), 2014 IEEE International Conference on*, 2014, pp. 884–889.

[13] D. Formby, S. S. Jung, J. Copeland, and R. Beyah, "An empirical study of tcp vulnerabilities in critical power system devices," in *Proceedings of the 2Nd Workshop on Smart Energy Grid Security*, ser. SEGS '14, Scottsdale, Arizona, USA: ACM, 2014, pp. 39–44, ISBN: 978-1-4503-3154-8.

[14] G. Shu and D. Lee, "Network protocol system fingerprinting - a formal approach," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, 2006, pp. 1–12.

[15] T. Kohno, A. Broido, and K. Claffy, "Remote physical device fingerprinting," *Dependable and Secure Computing, IEEE Transactions on*, vol. 2, no. 2, pp. 93–108, 2005.

[16] K. Gao, C. Corbett, and R. Beyah, "A passive approach to wireless device fingerprinting," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, 2010, pp. 383–392.

[17] J. Francois, H. Abdelnur, R. State, and O. Festor, "Ptf: passive temporal fingerprinting," in *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, 2011, pp. 289–296.

[18] S. Radhakrishnan, A. Uluagac, and R. Beyah, "Gtid: a technique for physical device and device type fingerprinting," *Dependable and Secure Computing, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.

[19] A. Bates, R. Leonard, H. Pruse, D. Lowd, and K. Butler, "Leveraging usb to establish host identity using commodity devices," in *Network and Distributed System Security (NDSS)*, ser. NDSS '14, San Diego, California, USA, 2014.

[20] O. Ureten and N. Serinken, "Wireless security through rf fingerprinting," *Electrical and Computer Engineering, Canadian Journal of*, vol. 32, no. 1, pp. 27–33, 2007.

[21] J. Verba and M. Milvich, "Idaho national laboratory supervisory control and data acquisition intrusion detection system (scada ids)," in *Technologies for Homeland Security, 2008 IEEE Conference on*, 2008, pp. 469–473.

[22] H. Lin, A. Slagell, C. Di Martino, Z. Kalbarczyk, and R. K. Iyer, "Adapting bro into scada: building a specification-based intrusion detection system for the dnp3 protocol," in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*, ser. CSIIRW '13, Oak Ridge, Tennessee, USA: ACM, 2013, 5:1–5:4, ISBN: 978-1-4503-1687-3.

[23] I. Fovino, A. Carcano, T. De Lacheze Murel, A. Trombetta, and M. Masera, "Modbus/dnp3 state-based intrusion detection system," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, 2010, pp. 729–736.

[24] A. Carcano, A. Coletta, M. Guglielmi, M. Masera, I. Fovino, and A. Trombetta, "A multidimensional critical state analysis for detecting intrusions in scada systems," *Industrial Informatics, IEEE Transactions on*, vol. 7, no. 2, pp. 179–186, 2011.

[25] O. Kosut, L. Jia, R. Thomas, and L. Tong, "Limiting false data attacks on power system state estimation," in *Information Sciences and Systems (CISS), 2010 44th Annual Conference on*, 2010, pp. 1–6.

[26] A. A. Cardenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry, "Attacks against process control systems: risk assessment, detection, and response," in *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, ser. ASIACCS '11, Hong Kong, China: ACM, 2011, pp. 355–366, ISBN: 978-1-4503-0564-8.

[27] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla, "Swatt: software-based attestation for embedded devices," in *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*, 2004, pp. 272–282.

[28] Y. Li, J. M. McCune, and A. Perrig, "Viper: verifying the integrity of peripherals' firmware," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11, Chicago, Illinois, USA: ACM, 2011, pp. 3–16, ISBN: 978-1-4503-0948-6.

[29] "Ieee standard for electric power systems communications – distributed network protocol (dnp3)," *IEEE Std 1815-2010*, pp. 1–775, 2010.

[30] A. B. Downey, "Lognormal and pareto distributions in the internet," *Computer Communications*, vol. 28, no. 7, pp. 790 –801, 2005.

[31] V. Jacobson, "Congestion avoidance and control," *SIGCOMM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, Aug. 1988.

[32] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, *Tcp selective acknowledgment options*, RFC 2018, 1996.

[33] V. Paxson and M. Allman, *Computing tcp's retransmission timer*, RFC 2988, 2000.

[34] V. Paxson, M. Allman, J. Chu, and M. Sargent, *Computing tcp's retransmission timer*, RFC 6298, 2011.

[35] I. Psaras and V. Tsaoussidis, "The tcp minimum rto revisited," in *IFIP Networking*, 2007.

[36] ICS-CERT, *Icsa-15-300-01*, https://ics-cert.us-cert.gov/advisories/ICSA-15-300-01, 2015.

[37] ——, *Icsa-15-295-01*, https://ics-cert.us-cert.gov/advisories/ICSA-15-295-01, 2015.

[38] A. Dunkels, "Full tcp/ip for 8-bit architectures," in *Proceedings of the 1st International Conference on Mobile Systems, Applications and Services*, ser. MobiSys '03, San Francisco, California: ACM, 2003, pp. 85–98.

[39] *Advisory (icsa-15-041-02)*, https://ics-cert.us-cert.gov/advisories/ICSA-15-041-02.

[40] L. Ljung, "Perspectives on system identification," *Annual Reviews in Control*, vol. 34, no. 1, pp. 1 –12, 2010.

[41] S. Sridhar, A. Hahn, and M. Govindarasu, "Cyber-physical system security for the electric power grid," *Proceedings of the IEEE*, vol. 100, no. 1, pp. 210–224, 2012.

[42] R. Langner, *To kill a centrifuge*, http://www.langner.com/en/wp-content/uploads/2013/11/To-kill-a-centrifuge.pdf, 2013.

[43] P. Srinivasan, Master's thesis, Georgia Institute of Technology, 2015.

[44] R. Kendrick *et al.*, *Visual encyclopedia of chemical engineering: valves*, http://encyclopedia.che.engin.umich.edu/Pages/TransportStorage/Valves/Valves.html.

[45] *Synchronous speed of electrical motors*, http://www.engineeringtoolbox.com/synchronous-motor-frequency-speed-d_649.html, The Engineering Toolbox.

[46] M. Peltola, *Ac induction motor slip what it is and how to minimize it*, http://www.powermation.com/eng_lib/ac_induction_motor_slip.pdf.

[47] A. Orlov, *Mahalanobis distance*, http://www.encyclopediaofmath.org/index.php?title=Mahalanobis_distance&oldid=17720, Encyclopedia of Mathematics.

[48] L. Watkins, W. Robinson, and R. Beyah, "A passive solution to the cpu resource discovery problem in cluster grid networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 22, no. 12, pp. 2000–2007, 2011.

[49] F. Valeur, G. Vigna, C. Kruegel, and R. Kemmerer, "Comprehensive approach to intrusion detection alert correlation," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 3, pp. 146–169, 2004.

[50] H. Debar and A. Wespi, "Aggregation and correlation of intrusion-detection alerts," in *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*, ser. RAID '00, London, UK, UK: Springer-Verlag, 2001, pp. 85–103, ISBN: 3-540-42702-3.

[51] *Nccic/ics-cert year in review fy 2015*, https://ics-cert.us-cert.gov/sites/default/files/Annual_Reports/Year_in_Review_FY2015_Final_S508C.pdf, 2016.

[52] D. Formby, *Out of control: ransomware for industrial control systems*, https://www.rsaconference.com/events/us17/agenda/sessions/7072-Ransomware.

[53] D. Formby, S. Durbha, and R. Beyah, *Out of control: ransomware for industrial control systems*, http://www.cap.gatech.edu/plcransomware.pdf.

[54] *140cpu65150 unity processor modicon quantum - 768 kb - 166 mhz*, http://www.schneider-electric.com/en/product/140CPU65150/unity-processor-modicon-quantum---768-kb---166-mhz/, Schneider Electric.

[55] *Analysis of the cyber attack on the ukrainian power grid*, http://www.nerc.com/pa/CI/ESISAC/Documents/E-ISAC_SANS_Ukraine_DUC_18Mar2016.pdf, E-ISAC, 2016.

[56] *Mr.plc: your automation help site*, https://www.mrplc.com/.

[57] *Micrologix 1100 programmable controllers: instruction set reference manual*, http://literature.rockwellautomation.com/idc/groups/literature/documents/rm/1763-rm001_-en-p.pdf, Allen Bradley.

[58] E. S. Page, "Continuous inspection schemes," *Biometrika*, vol. 41, no. 1/2, pp. 100–115, 1954.

[59] W. Diffie and M. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theor.*, vol. 22, no. 6, pp. 644–654, Sep. 2006.

[60] C. Castelluccia, A. Francillon, D. Perito, and C. Soriente, "On the difficulty of software-based attestation of embedded devices," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, ser. CCS '09, Chicago, Illinois, USA: ACM, 2009, pp. 400–409, ISBN: 978-1-60558-894-0.